
Facultad de Ciencias Económicas y Empresariales

# FORECASTER

## APLICACIÓN WEB SHINY PARA EL ANÁLISIS Y EXPERIMENTACIÓN DE MODELOS DE PREDICCIÓN DE SERIES TEMPORALES

**Autor:** Pablo Chaure Cordero
**Director:** José Portela González

MADRID | 2022

## RESUMEN

Forecaster es una aplicación web Shiny que asiste a los científicos de datos y analistas de negocio en el entrenamiento de modelos de predicción de series temporales sin necesidad de código. El usuario puede cargar cualquier set de datos de serie de tiempo sin importar la frecuencia en que estén los datos para generar predicciones de multitud de casos de uso. Forecaster es capaz de lograr esto con la integración de más de quince algoritmos y técnicas distintas con las que experimentar y generar modelos predictivos. Además, incorpora técnicas avanzadas de análisis estadístico de series de tiempo acompañadas de atractivas visualizaciones interactivas para captar todas las señales ocultas en los datos.



**Palabras clave:** *Aplicación web interactiva, análisis de series temporales, predicción de series temporales, machine learning, deep learning, Shiny, código abierto.*



## ABSTRACT

Forecaster is a Shiny web application that assists data scientists and business analysts in the training of forecasting models without the need to code. The user may upload any time series dataset no matter the frequency the underlying data is in to generate predictions for a wide range of use cases. Forecaster hosts more than fifteen different algorithms and techniques for model experimentation and generation. Moreover, it offers a broad selection of advanced statistical time series analysis techniques along with fancy interactive visualizations to help pick up on all the underlying signals in the data.



**Keywords:** *Interactive web application, time series analysis, time series forecasting, Shiny, machine learning, deep learning, open source.*

## ÍNDICE DE TEMAS

## ÍNDICE DE FIGURAS

## ÍNDICE DE TABLAS

## LISTADO DE ABREVIATURAS

| | |
|---|---|
| **API** | *Application Programming Interface* |
| **ARIMA** | *AutoRegressive Integrated Moving Average* |
| **CRAN** | *Comprehensive R Archive Network* |
| **DL** | *Deep Learning* |
| **ETS** | *Exponential Smoothing* |
| **IA** | Inteligencia Artificial |
| **IdC** | Internet de las Cosas |
| **IDE** | *Integrated Development Environment* o entorno de desarrollo integrado |
| **MA** | *Moving Average* |
| **ML** | *Machine Learning* |
| **PaaS** | *Platform as a Service* o plataforma como servicio |
| **RNN** | *Recurrent Neural Network* |
| **UI** | *User Interface* o interfaz de usuario |
| **UX** | *User eXperience* o experiencia de usuario |

# 1. INTRODUCCIÓN

## 1.1. Contexto del tema

Desde los inicios de la historia hasta nuestros días, el ser humano ha sentido la necesidad de anticiparse a los acontecimientos a través de la predicción del futuro. Esta actividad no siempre ha sido cuantitativa como lo es eminentemente hoy en día, sino que también ha estado inspirada en fuerzas naturales o sobrenaturales; y hasta se ha considerado como una actividad criminal y oscura en ciertos momentos de la historia.

En el siglo VII a.C. en Mesopotamia, los pronosticadores y adivinos se basaban en los gusanos del hígado podrido de una oveja para predecir el futuro. Alrededor de esa misma época histórica, la gente que deseaba conocer el futuro viajaba a Delfos en Grecia para consultar al oráculo que realizaba sus predicciones mientras estaba intoxicado con vapores que surgían de la tierra. Los pronosticadores lo tuvieron más difícil bajo el emperador Constancio, que promulgó un decreto en el 357 d. C. que prohibía que "Cualquiera que consulte a un adivino para adivinar el futuro sufrirá la pena capital". Una prohibición similar de las predicciones se produjo en Inglaterra en 1736, cuando se convirtió en delito cobrar dinero por emitir predicciones y el castigo era de tres meses de prisión con trabajos forzados. Se pueden llegar a entender estas penas debido al poder que una predicción puede llegar a tener: un pronóstico preciso que se cumple puede parecer mágico, mientras que uno alejado de la realidad puede llegar a ser peligroso.

Avanzando hasta nuestro tiempo en el siglo XXI, la actividad de predicción del futuro no es, por suerte, un delito ni está basada en técnicas mágicas o sobrenaturales (obviando métodos menos científicos como la astrología u otras pseudociencias). En nuestro caso, la predicción del futuro se concreta sobre todo en la predicción de datos históricos de series de tiempo.

Los directivos y empresarios responsables de la toma de decisiones de las empresas y los dirigentes públicos encargados de las políticas gubernamentales hoy en día dependen de análisis cuantitativos y previsiones económicas para orientarse en el mundo competitivo en el que operan. Esto, unido a la globalización de los negocios, ha hecho que la capacidad

de anticiparse a los eventos sea una competencia importante para el buen hacer de las empresas y del sector público.

El objetivo de la predicción de series temporales o *forecasting* es proporcionar a los directivos información que asista en su proceso de toma de decisiones. Prácticamente todas las organizaciones, públicas o privadas, operan en un entorno incierto y dinámico y poseen un conocimiento imperfecto acerca del futuro. La tarea de previsión es una parte fundamental de cualquier sistema de planificación y control, y cualquier organización necesita de un procedimiento tal que le permita predecir el futuro de forma eficaz y apropiada para su actividad.

Gran parte del éxito empresarial proviene de la capacidad de prever oportunidades o contingencias futuras y de tomar las decisiones correctas anticipándose a ellas. El *forecasting* puede utilizarse como herramienta para orientar esas decisiones empresariales, aunque siga existiendo cierto nivel de incertidumbre. A través de la combinación de análisis estadísticos y de conocimiento de campo, los encargados de la tarea de predicción pueden reducir el nivel de incertidumbre que rodea a una decisión empresarial, aunque nunca podrán asegurar lo que ocurrirá en el futuro.

Como se puede intuir, este interés por anticiparse a hechos futuros es evidente y trae consigo un gran valor económico. Por este motivo, en las últimas décadas se han realizado numerosos avances en técnicas para predecir datos históricos que se han visto acelerados por el desarrollo de la ciencia de datos y el aprendizaje automático aplicados al *big data*. De este modo, una herramienta capaz de integrar la tecnología más innovadora y precisa sin necesidad de un conocimiento avanzado de programación resulta de gran relevancia para informar la toma de decisiones. Esta es precisamente la propuesta de valor que pretende abordar Forecaster.

## 1.2. Metodología

Para el desarrollo de Forecaster se ha seguido una metodología que combina distintos marcos muy habituales en el desarrollo de software, cada uno afectando a una parte diferente del proceso.

En primer lugar, se han seguido principios de la metodología *Agile* para la gestión del proyecto y la trazabilidad del trabajo. Se trata de una metodología muy extensa y que no aplica en su totalidad a un proyecto de este calado, pero se han tomado algunos de sus principios muy en cuenta. Trabajar así supone seguir un proceso iterativo y con la intención de emitir progresos y desarrollos en incrementos pequeños, pero asumibles. Así, se establecieron *sprints* de dos semanas en los que en cada uno se pretendía desplegar al menos una funcionalidad o parte de la aplicación. Los resultados se evalúan con un alto estándar de calidad técnica y diseño. Para la visualización del flujo de trabajo se ha utilizado un *backlog* recopilado en un tablero Kanban que incluye el estado de las tareas pendientes, activas, terminadas y bloqueadas.

Por otra parte, la metodología DevOps se ha seguido en la medida en que, al reunirse en la misma persona el desarrollo del software y los despliegues de este, la continua integración y el continuo despliegue de los cambios se han efectuado de manera conjunta y de un modo eficiente.

Finalmente, se han utilizado Git y Github muy extensamente para la trazabilidad de los cambios producidos durante el desarrollo. Estas herramientas permiten tener visibilidad de todos los cambios, mantener la gobernanza sobre el código y la posibilidad de publicarlo en un repositorio remoto. Se ha dependido ampliamente de una estructura de ramas para el repositorio. Más concretamente, el proyecto se ha servido del llamado *feature branching*. Esto consiste en la diferenciación de ramas para cada una de los *features* o características en desarrollo. De este modo, existe una rama principal *master* en la que está el código definitivo y distintas *features* con código en progreso que, una vez se finaliza, se integra en la rama principal. Esto permite compartimentar el código y poder saltar de una versión a otra sin afectar al resultado estable y en producción.

### 1.3. Objetivo del trabajo

El presente Trabajo de Fin de Grado trata de analizar el estado de la cuestión de las aplicaciones web para la predicción y análisis de series temporales para que sirva como fundamento en el desarrollo de Forecaster. En segundo lugar, pretende servir como

memoria del proceso de producción y desarrollo de todas las técnicas y motores embebidos dentro del software. Por último, este documento interesará como documentación técnica de la aplicación para el usuario final de la misma y para los futuros desarrollos que terceras personas quieran realizar sobre este proyecto de código abierto u *open source* (accesible a través del repositorio público de GitHub).

Por su parte, el fin de la herramienta Forecaster es el análisis, la visualización y la predicción de series temporales sin importar la realidad a que hagan referencia los datos. De este modo, el software sirve como un medio flexible para la experimentación y comparación de multitud de modelos y la selección del algoritmo más apropiado a la serie temporal sobre la que el usuario quiera trabajar.

La aplicación está diseñada para poder ser utilizada por personas del ámbito de la empresa (sin importar el departamento) con un perfil funcional y sin conocimientos avanzados sobre series temporales y lenguajes de programación. Partiendo de esa premisa para su diseño, todas las funcionalidades de Forecaster tienen la posibilidad de ser automatizadas a través de la interfaz de usuario o *User Interface* (en adelante UI, por sus siglas en inglés). De este modo, a través de los motores de cálculo incluidos, el usuario es capaz de obtener resultados de muy alto nivel sin ser preciso (aunque sí beneficioso) que disponga de competencias específicas de modelado y análisis de series temporales.

### 1.4. Acceso a Forecaster y al código

Forecaster está disponible de modo online a través de cualquier navegador en la siguiente dirección url: pablochaure.shinyapps.io/Forecaster. El código fuente en su versión más actualizada está disponible en el repositorio de GitHub: github.com/pablochaure/Forecaster.

## 1.5. Estructura del trabajo

El trabajo se estructura en cinco secciones principales:

En primer lugar, la introducción establece el contexto en que el trabajo se enmarca, así como los objetivos que pretende abordar y la metodología empleada para lograrlos. Además, se incluye una sección en la que se referencia cómo se puede acceder a Forecaster y al código fuente de la aplicación.

A continuación, se define el marco teórico con los conceptos fundamentales sobre los que versa el trabajo. Así, Forecaster se descompone en tres conceptos que quedan desarrollados en epígrafes separados. Se tratan nociones sobre aplicaciones web en general, se ofrecen detalles de Shiny como *framework* de desarrollo de aplicaciones web y, por último, se examina la cuestión del análisis y la predicción de series temporales.

En tercer lugar, se presenta un estudio comparativo de distintas herramientas de software y aplicaciones web que abordan la misma tarea que Forecaster. Se han analizado productos tanto propietarios como de tecnología *open source* y se han puesto en perspectiva las oportunidades y debilidades de Forecaster con respecto a estos competidores.

Seguidamente, se aborda la sección más extensa del trabajo que se ocupa de describir el desarrollo de Forecaster. Se detallan temas como el valor que aporta la aplicación, a quién va dirigido su uso y la arquitectura sobre la que está desarrollada. Esta sección hace también las veces de guía de usuario al exponer el flujo de trabajo sobre el que la aplicación está diseñada y los distintos componentes disponibles para su empleo.

Finalmente, la última sección establece las conclusiones del trabajo y ciertas reflexiones sobre el futuro de Forecaster.

# 2. MARCO TEÓRICO

En esta sección, se definen conceptos fundamentales para entender las distintas áreas de conocimiento tratadas en torno al proyecto. Forecaster se define, de modo conciso, como una aplicación web Shiny para el análisis y la experimentación de modelos de predicción de series temporales. Para entender mejor el proceso de desarrollo de Forecaster, esta definición se ha descompuesto en tres apartados.

En primer lugar, se tratan nociones fundamentales acerca de las aplicaciones web, su arquitectura y su conectividad. Una aplicación web es el formato que se ha escogido para presentar el trabajo técnico realizado. Por ello, resulta de interés entender ciertas características y el proceso formal de diseño que se ha seguido para su planteamiento, así como las herramientas para llevarlo a cabo.

En segundo lugar y relacionado con lo anterior, se define el *framework* de desarrollo de aplicaciones web Shiny. Se trata brevemente la reactividad en que se fundamenta este marco, la estructura de una *app* Shiny y los lenguajes que tiene embebidos para cada una de las partes de la arquitectura de la aplicación.

Por último, se introducen los conceptos básicos sobre series temporales como conjunto de dato y la predicción de estas en un horizonte futuro. Además, se detallan brevemente algunas de las técnicas gráficas para el análisis de series temporales, así como la tipología de modelos de *forecasting* existentes y los distintos métodos en que estos se clasifican.

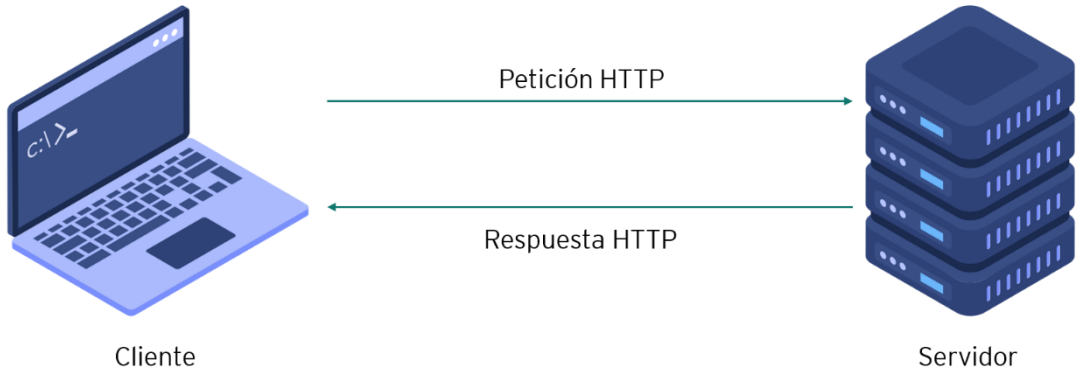
## 2.1. Aplicaciones web

Una aplicación web, también conocida con el término *web app*, designa a cualquier programa que es accesible a través de una conexión de red en lugar de estar instalada en la memoria local del dispositivo del usuario. Se tratan de programas diseñados y programados específicamente para ejecutar tareas concretas y cuya ejecución y computación se lleva a cabo en un servidor web (Luján-Mora, 2002).

### *2.1.1. Accesibilidad*

La conexión de una aplicación web habitualmente se realiza a través de la utilización del protocolo HTTP que sirve para poner en contacto las peticiones del ordenador cliente (usuario del programa) y las respuestas que el ordenador que tiene la información (el alojado en el servidor) da a la tarea solicitada (Anónimo, 2017). Se entiende el protocolo HTTP como el modo en que se comunican los ordenadores de la *World Wide Web* entre sí (Gourley et al., 2002).

**Figura 1:** Protocolo HTTP




*Fuente: elaboración propia*

El modo en que el usuario accede al servidor web donde está alojada la aplicación y puede ejecutar la tarea es por medio de un navegador web y a través de una red de internet o de intranet. De este modo, el navegador web serviría como medio donde se reproduce de forma gráfica el contenido del programa informático y la red se encargaría de transmitir la información computada en el servidor.

### *2.1.2. Arquitectura*

Es en el servidor donde se realiza todo el procesamiento de las tareas determinadas en el programa informático. Además, este ha de ser configurado de tal modo que cuente con las especificaciones técnicas necesarias para que ese procesamiento se ejecute con la capacidad suficiente. Esta capacidad vendrá determinada por los cálculos y procesos que han de ejecutarse para dar respuesta a las tareas para las que la aplicación ha sido diseñada (Escalona y Koch, 2002). Adicionalmente, el número de usuarios haciendo uso del

sistema y la carga de trabajo que estos demanden serán otros parámetros muy importantes para determinar los requerimientos del sistema de la máquina donde se ejecuta el programa.

Es precisamente esta última idea sobre la multitud de usuarios simultáneos y su acceso flexible al servicio en cuestión lo que supone una gran ventaja para las aplicaciones web accesibles desde internet. El hecho de que den servicio a una gran afluencia de usuarios desde un servidor implica que el mantenimiento y la actualización de la aplicación resulta mucho más sencillo. Estando ubicada en un servidor, el desarrollador es capaz de realizar correcciones y mejoras sin tener que distribuir e instalar de nuevo el software en cada una de las máquinas de los usuarios. Esto acorta el plazo de lanzamiento o *Time to Market* y permite un mayor uso efectivo de la aplicación.

Por otra parte, que concurran múltiples usuarios no supone inconveniente para su correcto funcionamiento. Es posible orquestar el servidor de tal modo que se ejecuten procesos de forma paralela para cada uno de los clientes que acceden al servidor. Esto vendrá determinado por el diseño de la arquitectura del sistema. No obstante, si no se optase por esta arquitectura paralela, se podrían atender todas las peticiones siempre y cuando los mencionados requerimientos del servidor sean los adecuados para soportar la carga de trabajo demandada en un momento determinado.

Adicionalmente, esta arquitectura tiene la ventaja de ser portable. Esto quiere decir que una aplicación es accesible y se podría ejecutar desde diferentes plataformas (sistemas operativos, clases de dispositivos, etc.) sin tener que particularizar el software. De este modo, una aplicación web podría ejecutarse en una máquina Windows, MacOS o Linux; en un dispositivo móvil o en una televisión inteligente (Alegsa, 2018).

No obstante a todo lo anterior, una aplicación web también puede ser ejecutada de modo local. Esto implica que toda la computación se realiza a través de la máquina del usuario si su contenido no ha sido subido a un servidor alojado en una máquina virtual o si el desarrollador así lo ha determinado. La dirección a través de la que se conecta el usuario a la aplicación se denomina dirección de *loopback* o *localhost* y no es más que un modo de referenciar al ordenador dentro del dominio de las redes (Bustos, 2021). Esta práctica es habitual que se produzca durante el proceso de desarrollo del programa para acortar el

tiempo que el desarrollador emplea en testear si el resultado de su trabajo es el deseado o si, por el contrario, ha de realizar modificaciones para llegar al producto final.

### *2.1.3. Arquitectura de tres niveles*

En cuanto a cómo se estructura una aplicación web ya se han adelantado algunos conceptos. Existe una arquitectura de software de aplicación habitual denominada *three-tier* o de tres niveles. Como su nombre indica, esta arquitectura organiza la aplicación en tres niveles (*tiers*) de computación distintos: nivel de presentación, nivel de aplicación y nivel de datos. Cada uno de estos niveles se desarrolla con herramientas, lenguajes e infraestructuras distintas y de modo independiente. A través de la combinación de los tres se llega al resultado definitivo que es la aplicación.

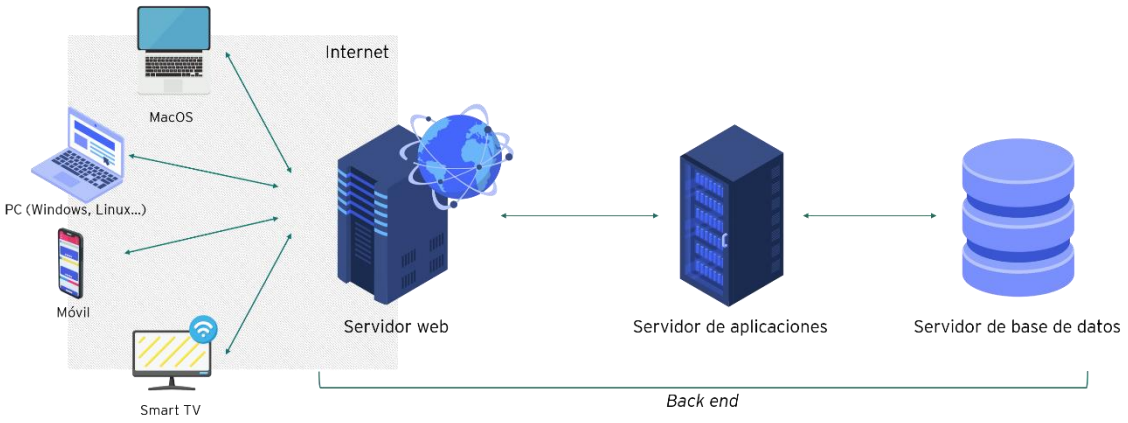
El nivel de presentación también es conocido comúnmente como el *front end* de una aplicación y, para el caso de una aplicación web, es representado por el servidor web. Se trata de la representación que se hace en el lado del cliente y es el nivel a través del cual el usuario es capaz de interactuar con la aplicación. Conforma la UI y la experiencia de usuario (UX, por las siglas de *User Experience* en inglés). La UI se encarga de mostrar la información, mientras que la UX se encarga de que la experiencia que el usuario tiene con la aplicación sea relevante, intuitiva y adecuada para la tarea (Anónimo, n.d.). El *front end* no solo muestra, sino que también recoge información del usuario como *input* para el funcionamiento de la aplicación. Este nivel se desarrolla fundamentalmente con HTML, CSS y JavaScript como lenguajes de programación para la estructura, el estilo y la interactividad, respectivamente.

El nivel de aplicación es el centro de la web app, la parte no visible responsable de todas sus funcionalidades. Es todo aquello que tiene lugar en la parte de los servidores de aplicaciones. Toda la información que ha sido recogida en el nivel de presentación es procesada en este segundo nivel a partir de una lógica programática o de negocio. Esta parte del programa busca obtener un resultado determinado para la función a que está destinada la aplicación y mostrárselo al usuario a través de la UI. De este modo, todo el intercambio de información que tiene lugar en la aplicación pasa por este nivel ya que los niveles de datos y de presentación no son capaces de comunicarse directamente. Los

lenguajes de programación más frecuentes en el desarrollo del nivel lógico son Python, Ruby, Java, PHP y Perl, sin ser esta lista exhaustiva.

Por último, el nivel de datos o *back end* es donde se almacena y gestiona toda la información de una aplicación (servidor de base de datos). Esto incluye tanto los *inputs* proporcionados por el usuario, como los *outputs* generados a través de los motores de cálculo que incluye la aplicación. El nivel de aplicación se comunica con este a través de llamadas a las API (del inglés *Application Programming Interface*). Pueden tratarse de servidores de bases de datos relacionales (SQL) como MySQL y PostgreSQL o de un sistema de gestión de bases de datos no relacionales (NoSQL) como MongoDB o CouchDB (Anónimo, 2021).

**Figura 2:** Arquitectura de una aplicación web de tres niveles




*Fuente: elaboración propia*

### *2.1.4. Frameworks de desarrollo web*

Como se ha ido mencionado, existen numerosas tecnologías, técnicas y lenguajes que se han de conocer para poder llevar a cabo de principio a fin el proceso de creación de una aplicación web. Estos son conocimientos que un desarrollador *full-stack* debe tener, pero que, individualmente, son difíciles de adquirir. Es por este motivo que existen los *frameworks* de desarrollo web.

La mayoría de los *frameworks* existentes tienen un objetivo en común: facilitar la tarea del desarrollo en un lenguaje determinado cuando hay muchos componentes que tratar

(Mauricio, 2018). Estos ofrecen una forma estandarizada y accesible de construir y desarrollar aplicaciones web. Al tratarse de paquetes o librerías creadas con su fundamento en la programación orientada a objetos, contienen distintas clases preprogramadas que son reutilizables y adaptables al software que se quiere desarrollar. De este modo, estas librerías proporcionan plantillas y patrones básicos para construir aplicaciones web fiables, escalables y que se puedan mantener en el tiempo. En otras palabras, un *framework* se puede concebir como una forma de crear atajos que pueden evitar que la tarea se convierta en abrumadora y repetitiva.

Existen numerosos *frameworks* de desarrollo como Django, Node.js y Angular. La elección de uno frente a otro dependerá fundamentalmente del lenguaje subyacente en el que se quiere desarrollar la web app. Por ejemplo, Django es un *framework* muy utilizado para el desarrollo de aplicaciones web con Python. Otros factores a tener en cuenta en el momento de la elección incluyen, entre otros, con qué tipo de base de datos se va a interactuar, la variedad y customización de los objetos HTML que se quiere crear y servicios adicionales que se quieren incluir en el funcionamiento (gestión de usuarios, por ejemplo) (Anónimo, 2015).

## 2.2. Shiny

Shiny es un paquete *open source* de código de R que proporciona un *framework* para desarrollar aplicaciones web interactivas utilizando R. Las aplicaciones Shiny son un medio muy útil y práctico para comunicar información de modo efectivo a través de elementos interactivos en lugar de documentos estáticos (Anónimo, n.d.). Una aplicación Shiny se compone de la UI por medio de un objeto `ui` y una función `server()` que contiene las instrucciones necesarias para construir los objetos y *outputs* que se muestran en la UI. La interacción del usuario con los elementos creados es posible gracias al modelo de programación reactiva que utiliza Shiny. De este modo, los elementos de la *app* se actualizan cada vez que el usuario modifica alguno de los *inputs*. Una gran ventaja que posee este *framework* es que, para crear aplicaciones Shiny, no es necesario tener experiencia alguna en desarrollo web. No obstante, se pueden conseguir resultados más

flexibles y personalizados mediante el uso de elementos y comandos de HTML, CSS y JavaScript.

### *2.2.1. Estructura de una aplicación Shiny*

Una aplicación Shiny puede ser creada de dos formas distintas: con un único archivo (*app.R*) o con dos (*ui.R* y *server.R*). Como se verá a continuación, la diferencia en la estructura de estas dos formas es mínima.

Una aplicación Shiny con un único archivo debe contener en el directorio del proyecto un *script* de R llamado, necesariamente, *app.R*. Este *script* incluirá tres partes diferentes: un objeto de UI que controla el diseño y la apariencia de la aplicación (`ui`), una función `server()` que contiene los comandos con las reglas lógicas y funcionales para generar los *outputs*, y una llamada a la función `shinyApp(ui = ui, server = server)` que genera la aplicación a partir de los parámetros `ui` y `server`.

La otra alternativa consiste en escribir dos archivos separados *ui.R* y *server.R* que contengan el objeto de la UI y la función del servidor, respectivamente (Chang, 2017). A continuación, la aplicación se genera llamando a `runApp("app_path")` donde `app_path` es la ruta del directorio donde se encuentran los archivos *ui.R* y *server.R*. Crear una aplicación con esta estructura diferenciada puede ser más apropiado para aplicaciones más extensas pues permite una gestión más fácil del código (Cetinkaya-Rundel, 2017).

### *2.2.2. Reactividad*

La reactividad es lo que hace que las aplicaciones Shiny sean interactivas. Esta característica permite que la aplicación se actualice instantáneamente cada vez que el usuario realiza un cambio en los *inputs* (Grolemund, 2015).

Existe una gran variedad de elementos de entrada o *inputs* y de elementos de salida o *outputs* que pueden mostrarse y generarse en una aplicación Shiny. Los *inputs* son elementos con los cuales el usuario es capaz de interactuar al modificar su valor inicial.

Los *outputs* son aquellos objetos que se quieren mostrar al usuario y que varían entre tablas de datos, visualizaciones, texto, gráficos, etc. Shiny utiliza un modelo de programación llamado reactividad para facilitar la interactividad de la aplicación y la interacción entre los valores de los *inputs* y los *outputs* que se generan (Moraga, 2019). Gracias a la reactividad, los valores de entrada se pueden modificar y automáticamente los *outputs* que toman dichos valores se actualizan. La reactividad elimina la necesidad de escribir código adicional para gestionar los eventos que se suceden en la aplicación (Anónimo, n.d.).

En Shiny, hay tres tipos de objetos que conforman la programación reactiva: fuentes (*sources*) reactivas, extremos (*endpoints*) reactivos y conductores (*conductors*) reactivos. La estructura más simple implica sólo una fuente y un extremo. La fuente suele ser el *input* del usuario a través de la UI del navegador. Por ejemplo, cuando el usuario selecciona una opción, escribe una entrada o hace clic en un botón, estas acciones establecen valores que son fuentes reactivas. Por otra parte, un extremo reactivo suele ser algo que se muestra en la ventana del navegador (un *output*), como un gráfico o una tabla. Es posible incluir componentes intermedios entre las fuentes y los extremos para controlar su computación y representación. Los conductores reactivos son utilizados habitualmente para encapsular operaciones lentas o de alto coste computacional (Anónimo, 2017).

## 2.3. Análisis y predicción de series temporales

Una serie temporal es una sucesión de datos u observaciones ordenadas cronológicamente. Ejemplos de series temporales son las visitas diarias a una página web, el consumo horario de gas de una fábrica y la tasa de desempleo mensual. Estos datos son observaciones de una característica concreta (serie temporal univariante) o de más de una característica (serie temporal multivariante) de una variable medible en distintos momentos temporales. Las observaciones están separadas por intervalos de tiempo regulares y uniformes, su frecuencia.

Según el análisis clásico, una serie temporal queda definida por cuatro componentes: tendencia, estacionalidad, variaciones cíclicas y el error o componente irregular.

- Tendencia (T): movimiento gradual de crecimiento o de decrecimiento a largo plazo.
- Estacionalidad (S): movimiento oscilatorio a corto plazo (menor a un año) sobre la tendencia que ocurre periódica y recurrentemente.
- Variación cíclica (C): movimiento oscilatorio superior al año en torno a la tendencia que ocurre regularmente, pero no recurrentemente.
- Error (E): la variación residual que resulta de la sustracción del resto de componentes. Suponen movimientos erráticos de la serie producidos por factores imprevisibles y que no atienden a ningún patrón.

De este modo, la serie se puede representar de modo esquemático de la siguiente forma:

- Ecuación aditiva: $Y_{it} = T_{it} + S_{it} + C_{it} + E_{it}$
- Ecuación multiplicativa: $Y_{it} = T_{it} * S_{it} * C_{it} * E_{it}$

### *2.3.1. ¿Qué se puede predecir?*

La predicción de series de tiempo es una actividad utilizada en numerosos casos de uso: para aprovisionar un supermercado de productos es necesario anticipar la cantidad de cada artículo que va a ser comprado; para establecer la cantidad de servidores de un servicio web se ha de determinar la carga que se espera soportar; para determinar cuántos vehículos eléctricos se han de fabricar hay que prever la demanda que habrá de estos en los próximos años.

Las predicciones se pueden necesitar con varios años de antelación (para el caso de inversiones de capital), o con sólo unos minutos de antelación (para el enrutamiento de telecomunicaciones). Sean cuales sean las circunstancias o el horizonte temporal, la previsión del futuro supone una ayuda muy importante para una planificación eficaz y eficiente. Como se ha mencionado, en el ámbito de la empresa o los negocios, este ejercicio aporta una gran base de información para la toma de decisiones.

Algunos casos son más fáciles de predecir que otros. La posibilidad de predecir un acontecimiento o un valor depende de varios factores, entre ellos: el grado de comprensión de los factores que influyen en ello, la cantidad de datos que se disponen y

si las mismas predicciones pueden afectar a aquello que se quiere predecir (Hyndman, R. J. y Athanasopoulos, 2018).

### *2.3.2. Métodos para predecir una variable temporal*

El método de predicción más adecuado para la tarea depende en gran medida de los datos que estén disponibles.

Si no existen datos o si los datos disponibles no son relevantes para la predicción que se quiere hacer, los métodos cualitativos resultan más apropiados. Estos métodos no son meras conjeturas, existen estrategias y planteamientos estructurados y adecuados para obtener buenas previsiones sin utilizar datos históricos. Dentro de esta categoría son conocidos el método Delphi, la previsión por analogía o las encuestas de mercado.

Por otra parte, cuando existe información numérica acerca del pasado y cuando se puede asumir que ciertos patrones del pasado van a continuar manifestándose en el futuro los métodos cuantitativos resultan mucho más adecuados. Es en estos métodos en los que Forecaster encuentra su razón de ser y más concretamente cuando se trata de datos de series temporales. Los métodos univariantes son los más sencillos para la previsión de series temporales, pues sólo tienen en cuenta información sobre la variable que se quiere predecir y no intentan descubrir los factores externos que afectan a su comportamiento. Por lo tanto, infieren la tendencia y los patrones estacionales de la serie, pero ignoran el resto de la información que la afecta (promociones, datos macroeconómicos, etc.). Por el contrario, los métodos multivariantes tienen en cuenta todas estas variables dependientes que afectan al comportamiento de la variable objetivo y saben captar el efecto que tienen sobre esta.

### *2.3.3. Gráficos para el análisis de series temporales*

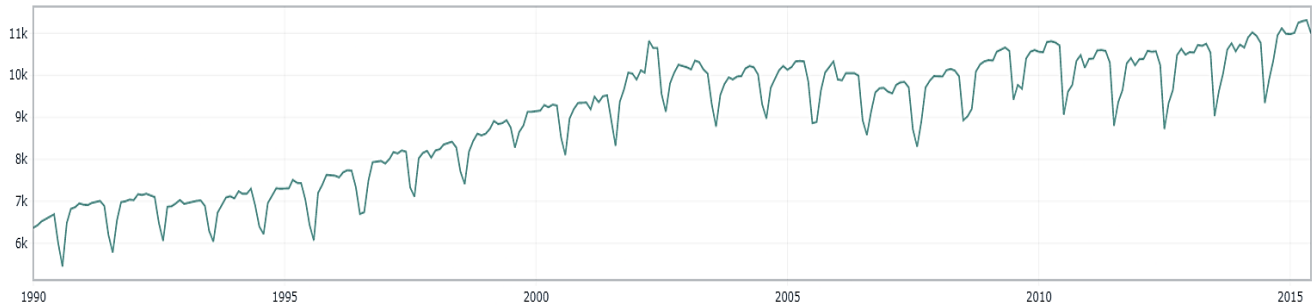
Para predecir adecuadamente una serie y escoger el modelo que más se ajusta a su comportamiento, un entendimiento preciso de la serie es fundamental. Este entendimiento es posible a partir del análisis y representación de todas las señales que definen la serie. En el ámbito de las series temporales se emplean numerosos métodos gráficos para llevar a cabo esta tarea. Los gráficos permiten visualizar muchas de las características de los

datos como patrones subyacentes, observaciones anómalas, cambios de tendencia en el tiempo y relaciones entre variables.

a. Gráfico de tiempo

El gráfico evidente con el que empezar este análisis es la visualización de la serie temporal. Consiste en representar las observaciones en función del momento de anotación y unir consecutivamente todos los puntos observados por una línea.

**Figura 3:** Gráfico de tiempo

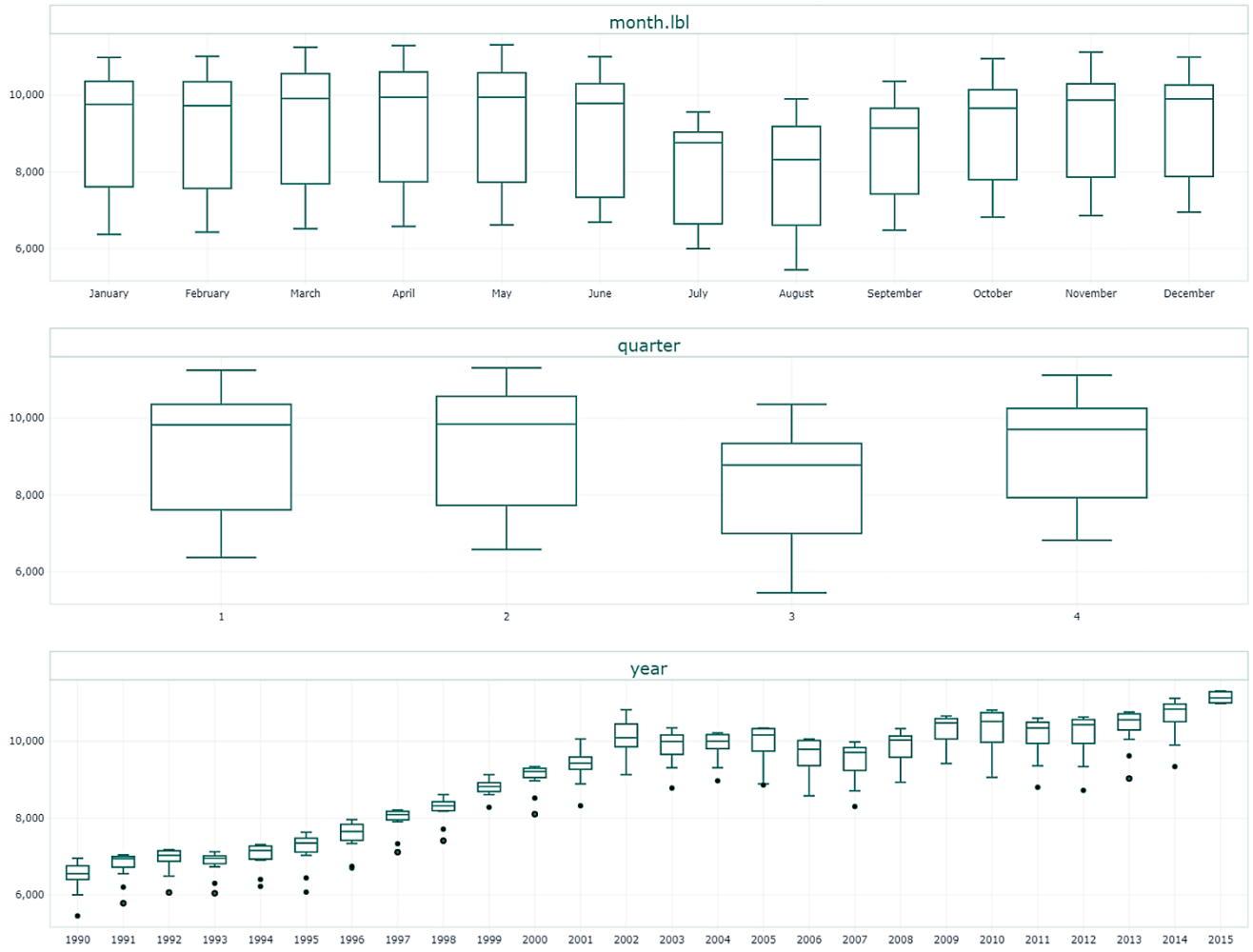



*Fuente: elaboración propia a partir de los resultados obtenidos a través de Forecaster*

b. Gráficos de estacionalidad

Un gráfico estacional es similar a un gráfico de tiempo, a excepción de que los datos se representan con respecto a las "estaciones" individuales en las que se observan los datos. Existen múltiples formas de representar la estacionalidad de la serie. En la Figura 4 se ha optado por su representación a través de *boxplots* por unidad de tiempo.

**Figura 4:** Gráficos de estacionalidad

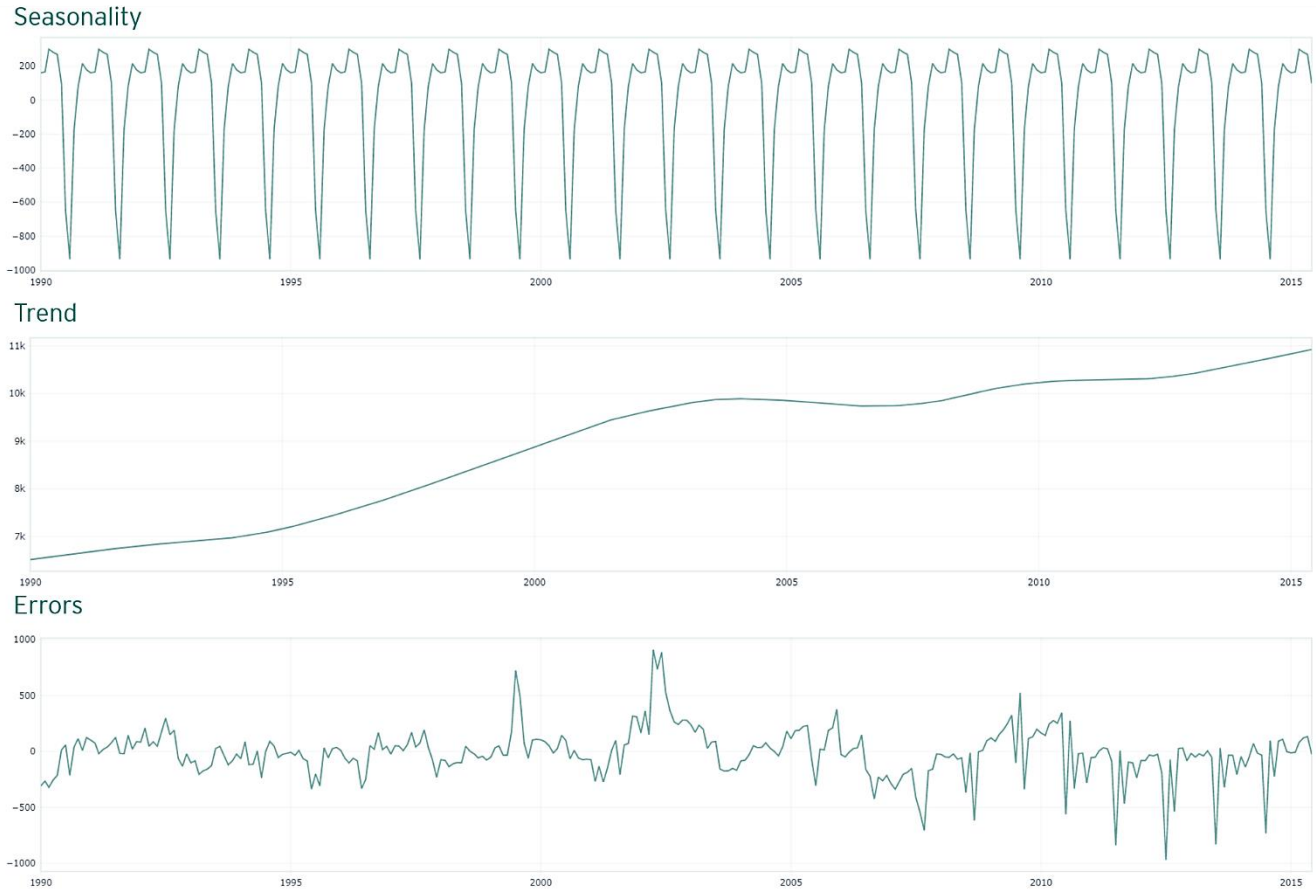



*Fuente: elaboración propia a partir de los resultados obtenidos a través de Forecaster*

c. Gráficos de descomposición de la serie

Como se ha explicado previamente, toda serie temporal se puede descomponer en ciertas características que esta presenta. Resulta conveniente representar gráficamente el patrón estacional, la tendencia y el error que, en conjunto, conforman la serie. Los gráficos individuales son a su vez series de tiempo también.

**Figura 5:** Gráficos de descomposición

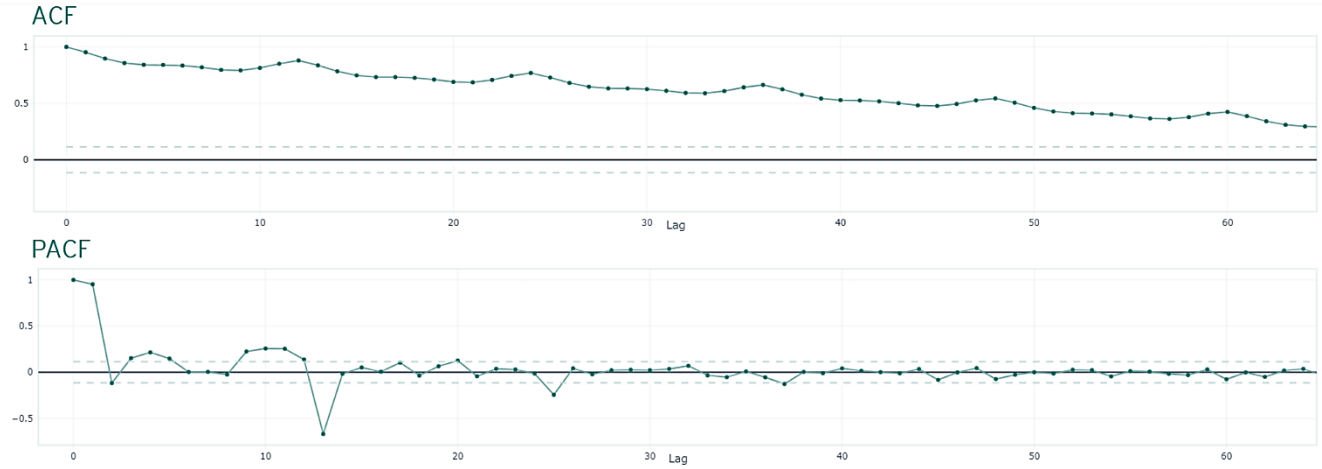



*Fuente: elaboración propia a partir de los resultados obtenidos a través de Forecaster*

d. Gráficos de autocorrelación o correlogramas

Del mismo modo que la correlación mide el grado de relación lineal entre dos variables, la autocorrelación mide la relación lineal entre los valores diferidos o *lags* de una serie temporal. En un gráfico de autocorrelación quedan representados los coeficientes que corresponden a cada relación entre el valor último y su *n-lag*. Otro de los elementos importantes que se muestra es el nivel de significación de los coeficientes.

**Figura 6:** Gráficos de autocorrelación

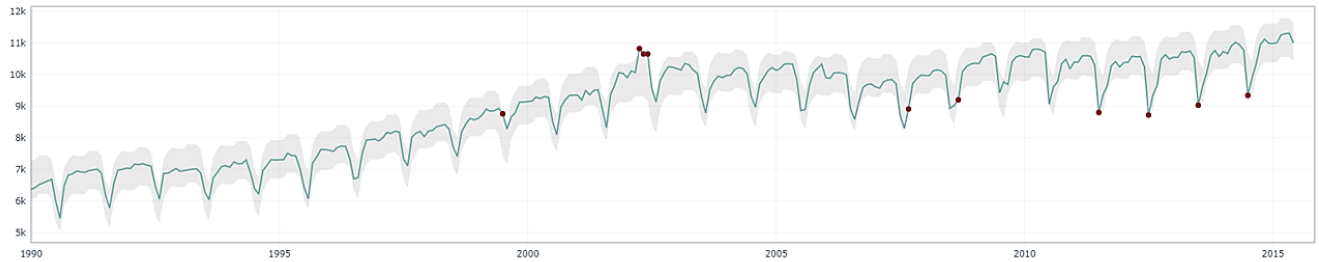



*Fuente: elaboración propia a partir de los resultados obtenidos a través de Forecaster*

e. Gráfico de anomalías

Una anomalía es una observación atípica que no sigue el patrón común colectivo o normalidad de la mayoría de los puntos de la serie y, por tanto, puede distinguirse fácilmente del resto de los datos. Para considerarlo como anomalía se suele recurrir al intervalo de confianza que presenta la serie y del cual los puntos anómalos se salen. Su representación es la misma que el gráfico de tiempo, pero con el añadido del intervalo de confianza y el destacado de los *outliers.*

**Figura 7:** Gráfico de anomalías




*Fuente: elaboración propia a partir de los resultados obtenidos a través de Forecaster*

### *2.3.4. Tipología de modelos de predicción*

La predicción de series temporales es una tarea compleja para la cual nunca podrá haber certeza absoluta de los resultados obtenidos. Esta tarea utiliza datos históricos y los patrones subyacentes para predecir la actividad futura de la serie. Se considera imposible conocer a priori qué técnica o modelo va a dar los valores futuros que van a acontecer. Como dijo George Box: "todos los modelos están equivocados, pero algunos son útiles" (Box, 1979). Es por eso que, la persona interesada en predecir el futuro de una serie temporal ha de conocer múltiples técnicas y algoritmos. Es habitual que en este proceso de predicción se experimente con diferentes técnicas para seleccionar aquella que parece que aporta los mejores resultados para las características de la serie.

No existe una taxonomía específica de algoritmos o técnicas para el entrenamiento de modelos de predicción de series temporales. No obstante, se han identificado tres grandes grupos de modelos que son los que Forecaster incluye.

En primer lugar, existen numerosos modelos clásicos o estadísticos que llevan poniéndose en práctica desde la década de 1970 cuando los modelos ARIMA consolidaron su utilidad. Además de los modelos de media móvil (MA, por las siglas en inglés de *Moving Average*), dentro de esta categoría también están los modelos de suavizado exponencial (ETS, del inglés *Exponential Trend Smoothing*) con el famoso algoritmo de Holt-Winters'; el modelo naive, regresión lineal y el modelo Croston para la predicción de demanda intermitente.

En segundo lugar, el aprendizaje automático o *Machine Learning* (ML, en adelante) ha aportado multitud de algoritmos que aplican a problemas de aprendizaje supervisado y no supervisado. El caso de la predicción de series temporales es clasificable como un problema de aprendizaje supervisado debido a que en los datos hay una o muchas variables de entrada que sirven para predecir una variable objetivo o de salida. Esto permite la aplicación de algoritmos de ML como XGBoost, árboles de decisión, Support Vector Machines, MARS y K-NN al problema de la predicción de series temporales.

Por último, en los últimos años se ha avanzado enormemente en el desarrollo de algoritmos de aprendizaje profundo o *Deep Learning* (DL, en adelante) para su aplicación en la predicción de series de tiempo. Con el mismo fundamento que para el caso de ML, los algoritmos de DL para problemas de aprendizaje supervisado resultan apropiados y

sorprendentemente potentes para realizar tareas de *forecasting*. Es tanto así que empresas como AWS han desarrollado librerías dedicadas únicamente al entrenamiento de modelos de DL para series temporales. Este es el caso de GluonTS que reúne algoritmos como DeepAR, N-Beats, Spacetimeformer, GP Forecaster y Deep State.

# 3. ANÁLISIS COMPARATIVO DE LAS ALTERNATIVAS A FORECASTER DISPONIBLES EN EL MERCADO

Tanto en el ecosistema de software *open source* como en el mercado de herramientas de tecnología propietaria desarrollada por empresas, existen alternativas a Forecaster para el análisis y la predicción de series temporales. Las distintas opciones disponibles tienen sus particulares ventajas y podrían llegar a ser preferibles para ciertos casos de uso frente a Forecaster. No obstante, estas también presentan puntos de mejora o deficiencias que suponen oportunidades para la herramienta presentada en este trabajo.

Se ha llevado a cabo un estudio comparativo de esta clase de herramientas de software equivalentes a Forecaster para analizar dónde se ubica dentro de las posibles elecciones. Las características que se han buscado en estos programas incluyen: "series temporales", "experimentación", "análisis", "exploración", "UI", "interactivo", "visualización", "*open source*", "*webapp*", "*dashboard"* y "automatización", entre otras tantas categorías en las que Forecaster se enmarca. Estos términos sirvieron en las búsquedas web para arrojar luz sobre qué oferta hay disponible y que esta estuviese acotada a cualidades similares a las de Forecaster. De entre los distintos programas de software se ha decidido seleccionar cinco alternativas tanto de código abierto, como de software patentado.

## 3.1. Amazon Forecast[1]

Amazon Web Services comercializa una herramienta de software propietario para las necesidades organizativas en el trabajo y tratamiento de series de tiempo. Se trata de una herramienta muy potente capaz de predecir series temporales jerárquicas, agrupadas e individuales de hasta 10.000 series. Al tratarse de Amazon el desarrollador, está pensada con el foco puesto en datos de inventario y operaciones para la previsión de demanda, sin excluir otros tantos casos de uso para los que también es apta.

Amazon Forecast utiliza ML y DL para la generación de predicciones de demanda sin necesidad de código y sin requerir conocimiento alguno sobre ML. Se promociona como

[1] Más información en la página oficial de AWS para este servicio: aws.amazon.com/forecast/

una herramienta que incluye la misma tecnología que Amazon utiliza en sus operaciones y que ha sido refinada con su experiencia en el sector. Se encuentran disponibles un total de seis algoritmos diferentes: Prophet, DeepAR+, CNN-QR, NPTS, ARIMA y ETS. Este servicio encuentra el mejor algoritmo para el caso concreto y, además, entrena un modelo ensamblado de todos los algoritmos generando así el mejor modelo posible.

Este servicio incluye funcionalidades realmente avanzadas y útiles como la capacidad de explicar cómo afectan los distintos factores individualmente a la predicción, generando modelos altamente explicativos. Un elemento particular en el que Amazon se basa para esta tarea es la inclusión de datos meteorológicos para informar las predicciones. De este modo, esta herramienta incluye de forma automática esta tipología de información para generar las predicciones probabilísticas finales. Una gran ventaja que ofrece es la integración sencilla con otras aplicaciones comunes para la gestión de negocio y las cadenas de distribución y producción.

### 3.2. Azure Time Series Insights[2]

Microsoft dentro de su conjunto de servicios de tecnología *cloud*, Azure, ofrece el servicio Azure Time Series Insights para el análisis, la visualización y la monitorización de datos de Internet de las Cosas (IdC, en adelante) obtenidos a través de sensores. Permite simplificar y agilizar el almacenamiento de datos de IdC en tiempo real para su análisis sin necesidad de escribir código.

Los datos de sensores suelen ser generados a frecuencias determinadas y vienen acompañados del momento en que se registra su medición, convirtiéndolos en datos de serie temporal. Dichos datos han de ser contextualizados a través de un modelo de IdC que logre captar el flujo de información de la instalación. Azure Time Series Insights permite una alta adaptación de tal modelo para las necesidades y los componentes específicos de la organización.

[2] Más información en la página oficial de Microsoft Azure para este servicio: azure.microsoft.com/en-us/services/time-series-insights/

La mayoría de las funcionalidades que incorpora son de tipo analítico más que predictivo. No obstante, integra en el sistema funcionalidades de ML para la detección de anomalías en los datos y la evaluación del comportamiento de los sensores. Permite ampliar enormemente sus funciones con la integración de otros productos de Azure para dispositivos de IdC.

Se trata de un servicio interactivo comercializado por una de las compañías más grandes e importantes de tecnología de nuestro tiempo. Por ello, el servicio es de código cerrado y no se distribuye al público de manera gratuita. Además, esta clase de producto solo interesará a individuos con necesidades de analizar y registrar sus datos de IdC cuando provienen de una instalación considerablemente compleja.

### 3.3. Arauto[3]

Arauto es un proyecto de *open source* diseñado para llevar a cabo la tarea de la predicción de datos históricos hacia el futuro. Se define como una herramienta para una ágil experimentación y entrenamiento de modelos de series de tiempo. Se basa en modelos estadísticos para proporcionar predicciones precisas que sirva para datos financieros, de ventas y algunos otros casos de uso tasados.

Arauto está construido sobre código de Python y distribuido a través de una app Streamlit. Se trata de una aplicación web interactiva que integra únicamente la generación de modelos ARIMA. A pesar de solo tener un modelo disponible, las posibilidades de análisis y ajuste del modelo específicas para ARIMA son muy completas y cubren todas las necesidades para entrenar modelos de esta clase. De este modo, se computan distintos tests estadísticos como la prueba de Dickie-Fuller aumentada y las funciones de autocorrelación y autocorrelación parcial de modo analítico y visual.

Algunas de las desventajas de Arauto es que no permite la subida de datos por parte del usuario, limitando bastante las posibilidades de uso y la flexibilidad que esto aporta. Contiene una muestra de cuatro conjuntos de datos entre los que optar para la

[3] Más información acerca de Arauto en: towardsdatascience.com/introducing-arauto-an-interactive-tool-for-time-series-forecasting-14b3e36b5f81 . Accesible a través de: projectarauto.herokuapp.com/

experimentación con ARIMA. A esto hay que añadir algún detalle como que las visualizaciones que se muestran no son interactivas y eso reduce la información que aporta. No obstante, una característica interesante y útil es que genera y muestra el código de Python con el que es posible replicar los resultados de los modelos generados a partir de los parámetros introducidos por el usuario.

### 3.4. Streamlit Prophet App[4]

La siguiente alternativa que se ha encontrado y analizado es la aplicación Streamlit Prophet. Como su nombre indica, se trata de una aplicación distribuida con Streamlit para la especificación de modelos a partir del algoritmo Prophet. Este software ha sido desarrollado por Facebook y se trata de un modelo aditivo que capta la tendencia, estacionalidad de la serie y el efecto que los festivos tienen en los datos. Su funcionamiento mejora con series que tengan patrones estacionales marcados y con múltiples ciclos de datos históricos.

Streamlit Prophet es una herramienta *open source* que asiste a los científicos de datos en el entrenamiento de modelos de *forecasting* sin la necesidad de código y que está disponible a través de su repositorio de GitHub. Permite cargar un conjunto de datos con valores históricos a partir de los cuales la app entrena un modelo predictivo en base a la definición de unos parámetros por el usuario. Genera diversas visualizaciones para evaluar su precisión y capacidades, y ofrecer información adicional para explicar el funcionamiento del modelo.

De entre las funcionalidades de la aplicación destaca la exploración minuciosa de los datos que se puede hacer. Además, al integrar únicamente Prophet, la aplicación permite obtener mucha información explicativa del comportamiento del modelo entrenado al estar específicamente diseñada para este algoritmo. De este modo, se puede hacer un diagnóstico minucioso de los residuos, un examen de las métricas de precisión por componente del modelo, la descomposición de cada uno de los elementos del modelo y cómo estos afectan al modelo. Esto es posible gracias al ajuste de todos los

[4] Más información en: blog.streamlit.io/forecasting-with-streamlit-prophet/ Accesible a través de: share.streamlit.io/maximelutel/streamlit_prophet/main/streamlit_prophet/app/dashboard.py

hiperparámetros de Prophet y a la inclusión de regresores externos a partir de otro archivo que permiten captar todas las señales posibles de los datos proporcionados.

### 3.5. Time Series Forecasting Shiny[5]

Esta última herramienta *open source* no tiene un nombre público concreto con la que designarla. Al igual que Forecaster se trata de una aplicación Shiny para la predicción de series temporales, específicamente diseñada para la previsión de demanda de frecuencia mensual. Permite la carga de datos por el usuario, aunque no presenta excesiva flexibilidad en esta parte ya que el archivo que se sube debe tener un formato determinado y que los datos se presenten con una configuración específica para que la aplicación sea capaz de cargarlos correctamente.

La aplicación Shiny ofrece la posibilidad de entrenar modelos estadísticos únicamente, concretamente: ETS, TBATS, ARIMA, Croston y regresión lineal. Estos modelos se pueden entrenar tanto de modo individual para todas las series presentes en el conjunto de datos, como en grupo (*batch*) para una única serie. En este último caso, la aplicación evalúa sus resultados y escoge aquel que presenta mejor precisión en función de una métrica escogida.

Por último, proporciona una sección de análisis estadístico en el que se muestran gráficos de descomposición de la serie, las funciones de autocorrelación total y parcial; y un gráfico de línea con la estacionalidad. Una ventaja que presenta es que todas las visualizaciones de la aplicación son interactivas y aumenta la granularidad de la información que el usuario puede obtener.

[5] Más información en: github.com/mlombera94/forecast_R-shiny. Accesible a través de: mlombera.shinyapps.io/forecast_r-shiny/

## 3.6. Síntesis de resultados y comparativa total de las herramientas

El estudio realizado se ha reducido a tres variables fundamentales que caracterizan las aplicaciones estudiadas y que son prueba de las posibilidades y el potencial que ofrecen. Estas tres variables son: el tipo de acceso a la herramienta (*open source* o propietario), la flexibilidad de casos de uso que se pueden tratar y las posibilidades de experimentación de modelos.

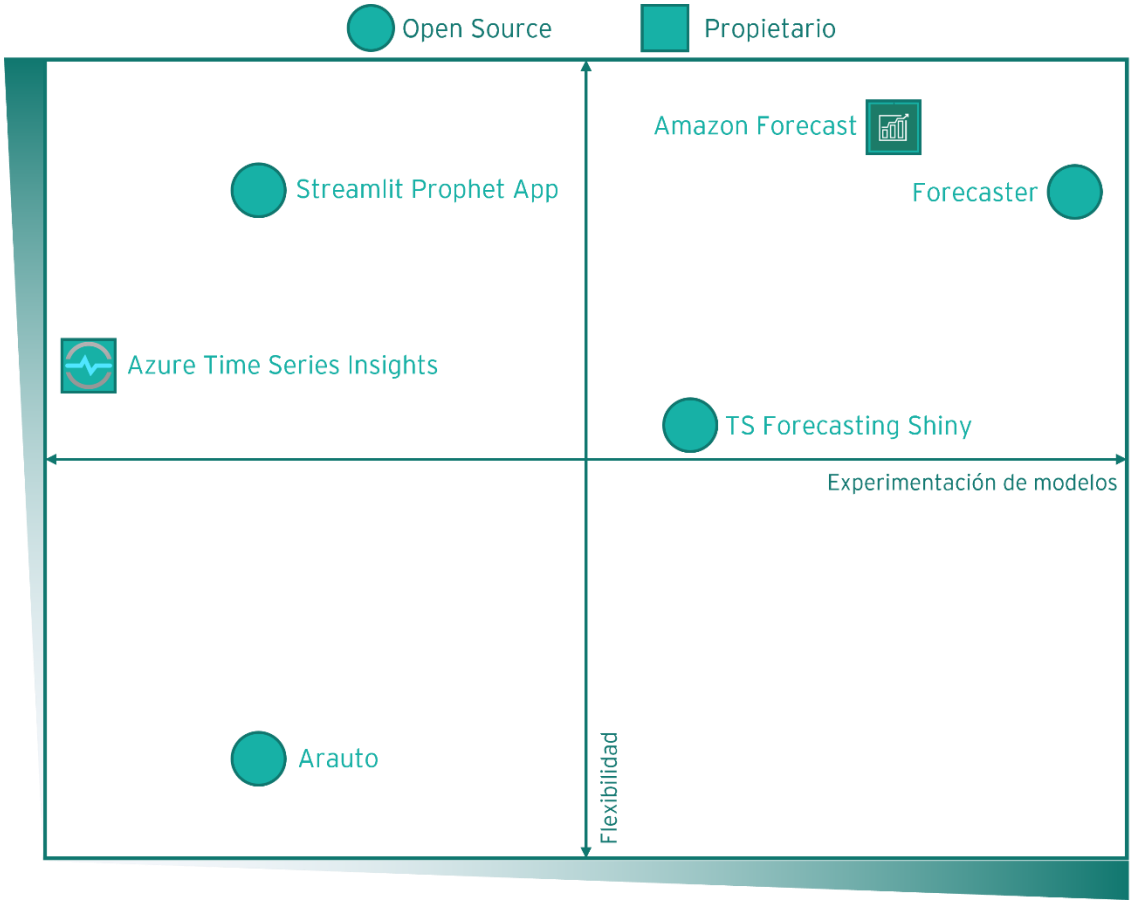
En la Figura 8 se muestran estas tres variables de modo gráfico quedando representados en los ejes horizontal y vertical la experimentación y la flexibilidad, respectivamente; y el acceso a cada herramienta queda ilustrado a través de la forma con que se referencia cada producto.

Este estudio no ha incluido ninguna herramienta que ofreciese buenas capacidades de modelado para pocos casos de uso y únicamente Arauto se sitúa alejada del resto en términos de las dos primeras variables. Por otro lado, el espectro de experimentación de modelos queda bastante bien representado teniendo claros líderes como Amazon Forecast y Forecaster y con el servicio de Azure a la cola en este respecto.

Forecaster se sitúa como una de las herramientas más prometedoras entre las alternativas estudiadas y descritas. Ofrece un alto grado de flexibilidad de casos de uso al permitir la carga de cualquier tipo de serie temporal que cumpla unos mínimos requisitos de formato, que otras como la aplicación de Prophet no presentan. Por otro lado, proporciona un medio de entrenamiento de modelos de predicción con una amplia variedad de posibilidades entre las que elegir, superando incluso a herramientas de pago como Amazon Forecast en el número de algoritmos disponibles. Este último aspecto acerca de su disponibilidad es otra gran ventaja ya que, de forma completamente gratuita, cualquier persona es capaz de acceder a todas sus funciones y colaborar en sus futuros desarrollos a través del repositorio de GitHub donde está almacenado todo el código.

En conclusión, Forecaster se presenta para cubrir algunas de las necesidades que en el ecosistema *open source* parece que todavía no se han tratado para el análisis y predicción de series temporales y que algunas herramientas de pago no hacen accesible a todo el mundo. Resulta novedoso y conveniente adentrarse en el desarrollo de una aplicación web con las capacidades que, aparentemente, no se encuentran disponibles en el mercado.

**Figura 8:** Gráfico comparativo de algunas herramientas disponibles para el tratamiento de series temporales en términos de: flexibilidad de casos de uso, posibilidades de modelado y accesibilidad




*Fuente: elaboración propia*

# 4. DESARROLLO DE FORECASTER

## 4.1. Introducción

La herramienta desarrollada para este proyecto es una aplicación web que proporciona una interfaz de usuario para el análisis y predicción de series temporales. Forecaster reúne multitud de herramientas para satisfacer la mayoría de las necesidades que un usuario en busca de proyectar sus datos a un momento futuro pueda requerir.

Forecaster está diseñada para poder ser utilizada por usuarios de distintos perfiles debido a las posibilidades de automatización que incluye. De este modo, todas las capacidades de la aplicación, junto con el conocimiento del usuario, suponen una propuesta de valor excelente para la integración de este sistema dentro de las distintas áreas funcionales que conforman la empresa.

Esta integración se ve facilitada por la arquitectura con la que está diseñado el software y por los motores de cálculo y análisis embebidos en él. Se ha construido y programado exclusivamente con los lenguajes de programación R, Python, HTML y CSS, además de multitud de *APIs* disponibles para cualquier desarrollador. Todos ellos se tratan de lenguajes, librerías, paquetes y módulos de código *open-source*. Además, se ha seguido un modo de trabajo basado en la metodología de desarrollo DevOps para la continua integración, desarrollo y despliegue de los avances realizados en el software. Como pieza fundamental de este modo de trabajo se ha utilizado Git, un sistema de gestión de código y de control de versiones, en conjunción con GitHub como servicio donde se almacena todo el código desarrollado.

Forecaster es accesible a través de un servicio de Internet gracias a que Forecaster y todos los componentes necesarios para su funcionamiento están instalados en una máquina virtual alojada en un servidor en la nube. Para esto se ha utilizado Shinyapps.io, la plataforma como servicio (PaaS) ofrecida por RStudio para albergar aplicaciones web Shiny. Además, también es accesible de modo local a través del código disponible en el repositorio público de GitHub.

La aplicación está diseñada con un proceso dividido en tres acciones: ingesta de los datos, análisis exploratorio y entrenamiento de los modelos de predicción. Se trata de un proceso flexible en cuanto a las posibilidades de modelado que ofrece: el usuario puede saltar de una metodología a otra diferente sin afectar a los resultados computados previamente (en el entrenamiento de los modelos o el análisis exploratorio).

Incluidas en Forecaster hay cuatro tipologías de modelos. Se incluyen modelos estadísticos o econométricos en uso desde hace décadas y técnicas más complejas y actuales que requieren un conocimiento más específico de conceptos de aprendizaje automático. De este modo, las metodologías que están implementadas en Forecaster son: ARIMA, ETS, *Machine Learning* y *Deep Learning.*

Durante el desarrollo de Forecaster se ha tenido en cuenta en todo momento los últimos avances en el campo de la predicción de series temporales. El objetivo que persigue es recopilar todas las técnicas más avanzadas que se han puesto en práctica en este ámbito de la ciencia de datos y aglutinarlas en una misma herramienta. El resultado presentado como parte de este trabajo es meramente una primera versión de lo que Forecaster puede llegar a ser y es por este motivo que el estado actual de la aplicación no logra cubrir completamente el objetivo mencionado. No obstante, sirve de motivación para continuar mejorando sus capacidades, integrándolas en el producto y desplegándolas para su aprovechamiento.

## 4.2. Propuesta de valor

Forecaster se ha desarrollado con la premisa inicial de crear una herramienta de experimentación de modelos para ser utilizada de modo flexible para cualquier caso de uso al que se quiera aplicar. Para lograr esto, una de las funcionalidades fundamentales que tiene Forecaster es la posibilidad de admitir sets de datos con formatos, frecuencias y estructuras muy diversas, dentro de ciertos límites. Esta característica supone uno de sus puntos fuertes por las posibilidades que permite, pero viene en detrimento de la capacidad de crear los modelos más ajustados a la serie temporal en cuestión. Por este motivo, Forecaster está destinada a ser una herramienta valiosa en los momentos iniciales del proceso de análisis y predicción de series temporales.

A través de las funcionalidades disponibles, el usuario es capaz de realizar un análisis exploratorio detallado para ir formándose una idea de cuáles son las características que la serie presenta. Por otro lado, con las cuatro metodologías de modelado y más de quince técnicas a disposición del usuario, este puede iterar entre todos ellos e ir analizando los resultados de cada uno. Es de utilidad para el usuario conocer tanto el mejor algoritmo para proyectar su serie, como la mejor combinación de los parámetros de cada algoritmo dadas las propiedades de la serie.

El usuario puede experimentar entre las distintas metodologías para generar modelos y evaluar cómo se comportan con los datos subidos. Para tomar una decisión sobre la metodología escogida, el usuario puede hacerlo a través de una inspección visual y analítica de los resultados del entrenamiento. Estos resultados se concretan en un conjunto cerrado de nueve métricas de precisión entre las que el usuario puede elegir y que son calculadas para cada modelo en el set de test.

Tras esta primera concreción de la metodología y el algoritmo que se va a utilizar, el usuario debe encontrar la mejorar configuración de este para la serie. Es posible modificar manualmente los parámetros escogidos de cada algoritmo para generar modelos con todas las combinaciones posibles de ellos. Así, podrá examinar a través de las métricas de precisión cómo se comporta cada uno de los modelos individualmente en la parte de entrenamiento y aproximar el modelo más adecuado.

El uso de una herramienta gráfica e interactiva como Forecaster es muy valioso para la tarea de previsión de series temporales debido a que permite obtener buenos resultados sin la necesidad de conocimiento alguno de lenguajes de programación por parte del usuario. El proceso detallado tiene la ventaja de facilitar y optimizar enormemente el proceso de creación de modelos para que, posteriormente con los resultados obtenidos, un usuario más experto pueda generar los modelos óptimos si se estima necesario. Esto supone una reducción significativa en la carga de trabajo en comparación con tener que generar cada modelo programática e individualmente.

Por último y en relación con lo anterior, la UI de Forecaster ha sido diseñada para tener una apariencia y estructura de *dashboard* habitual en herramientas de análisis de datos. Incorpora funcionalidades como la interactividad de todas las visualizaciones y tablas para mostrar la mayor información posible. Esto se realiza a través de los paquetes de R

*timetk* y *plotly*. Además, Forecaster incorpora la posibilidad de descargar los productos generados para su uso y aprovechamiento por otros sistemas del negocio.

### 4.3. Usuario objetivo

El diseño de la UX en el proceso de desarrollo de software es una parte crucial para el éxito que pueda tener un producto. Este es el proceso que utilizan los equipos de diseño para crear productos que proporcionen experiencias significativas y relevantes para los usuarios. Esto implica el diseño de todo el proceso de adquisición e integración del producto, incluidos los aspectos de marca, diseño, usabilidad y función (Anónimo, n.d.).

El foco principal de Forecaster no es tanto la UX como los modelos y productos que de ella se pueden obtener. No obstante, se han tenido muy en cuenta ciertos principios del diseño de UX para definir el empleo que se le va a dar y su configuración. En el caso que nos ocupa hacemos referencia a la definición de la audiencia objetivo para el producto.

En el momento de comenzar con el diseño y desarrollo de un producto digital es crucial definir no solo el uso que se le pretende dar, sino también estudiar quién va a ser el usuario a quien va dirigida la oferta. De este modo, definir correctamente la audiencia permite diseñar el producto para que esté en línea con el uso pretendido (Torres, 2018).

En el caso de Forecaster, se han identificado dos roles de negocio que pueden tener la necesidad específica de analizar y predecir series temporales en el desempeño de sus funciones. Es el caso de los analistas de negocio y los analistas/científicos de datos. Esta lista no es exhaustiva y sirve para analizar las capacidades y conocimientos que pueden tener los distintos roles sobre la materia para adecuar las funcionalidades de la aplicación.

Por un lado, los analistas de negocio tienen la función de utilizar los datos e información disponible para resolver los problemas de negocio que puedan darse. Se encargan de descomponer dichos problemas a través de identificar qué está pasando, cómo está pasando y por qué se hace lo que se hace desde un punto de vista funcional. Tienen altos conocimientos sobre empresa, finanzas y economía en general, pero competencias sobre programación y el procesamiento de datos no son su fuerte.

Por otro lado, los científicos y analistas de datos se encargan de extraer la información de los datos para crear valor para un negocio o una industria específica. Son capaces de lograrlo con su alto conocimiento sobre estadística, ciencias de la computación y el campo de conocimiento sobre el que trata la tarea (Finzer, 2013). A pesar de que es fundamental tener el conocimiento funcional para poder entender el problema, estas personas se suelen centrar más en los aspectos técnicos y la tecnología que se puede utilizar para resolverlos.

Resulta conveniente añadir a estas definiciones que ambos roles no serán los encargados de tomar las decisiones de negocio en base a los resultados obtenidos. Esa tarea corresponderá a cargos de mayor nivel como podría ser un CFO o un director de *marketing*. Estos tomarán los resultados del trabajo delegado en los analistas para informar las decisiones ejecutivas y estratégicas de negocio. Es por estos motivos que se los ha excluido de la audiencia objetivo, sin perjuicio de que puedan llegar a interactuar con la herramienta en algún momento.

En suma, existe una diferencia notable entre las competencias en el campo de la computación y el conocimiento de los distintos modelos entre los dos roles analizados. Este detalle no se ha pasado por alto en el desarrollo de Forecaster. Se podría decir que se han creado dos UX paralelas para cada uno de estos perfiles. Para un usuario menos conocedor de las características técnicas de los modelos la aplicación permite el entrenamiento de modelos completamente automatizados. Por otro lado, como ya se ha anticipado, existe la posibilidad de determinar los parámetros de los modelos de forma manual para aquellos usuarios más técnicos que conozcan la influencia que estos tienen en el modelo resultante.

## 4.4. Arquitectura del software

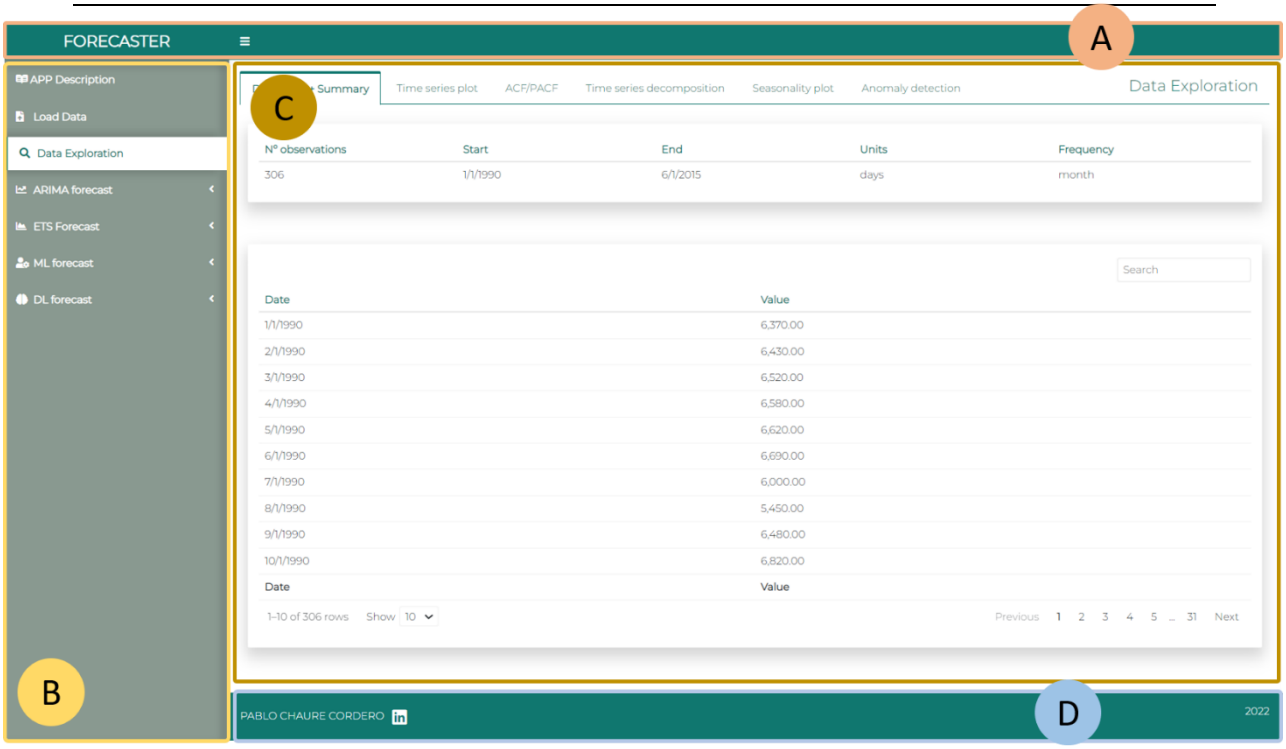
Como se ha mencionado en el epígrafe 2.2, una aplicación Shiny se conforma de dos partes necesariamente: la UI y el *server*.

### *4.4.1. UI*

a. Shinydashboard

La UI de Forecaster está diseñada con la estructura de un shinydashboard (Chang y Borges Ribeiro, 2018) como se puede apreciar en la Figura 9. Esta se compone necesariamente de (A) un `dashboardHeader()`, (B) un `dashboardSidebar()` y (C) un `dashboardBody()` para el encabezamiento (parte superior de la interfaz), la barra lateral situada en la izquierda de la pantalla y el resto de la pantalla o elemento principal, respectivamente. Adicionalmente Forecaster cuenta con (D) un `dashboardFooter()` o pie de página con los datos que me identifican como autor de la aplicación, un enlace a mi página de LinkedIn y el año en que esta se publicó.

**Figura 9:** Estructura de *dashboard* de Forecaster. **A)** Header. **B)** Sidebar. **C)** Body. **D)** Footer.




*Fuente: elaboración propia a partir de impresiones de pantalla de Forecaster*

Por su parte, el `dashboardBody()` tiene una estructura confeccionada con filas y columnas en las que se incluyen los diversos elementos HTML que muestran los *inputs* y *outputs* de la aplicación. Un elemento al que se ha recurrido en abundancia para

contener otros objetos es un `wellPanel()`. Este, por defecto, se trata de un panel con un borde y fondo gris, pero que en Forecaster se ha dado un estilo particular eliminando el borde, cambiando el fondo a color blanco y estableciendo una sombra para dotar a estos paneles de profundidad sobre el fondo del `dashboardBody()`.

El paquete dashboardthemes (Lilovski, 2021) también permite establecer un tema a nuestro shinydashboard. A través de la función `shinyDashboardThemes(theme = "poor_mans_flatly")` se ha aplicado el tema *Flatly* del *framework* de CSS Bootstrap. No obstante, este se ha modificado ligeramente para presentar la aplicación con colores diferentes.

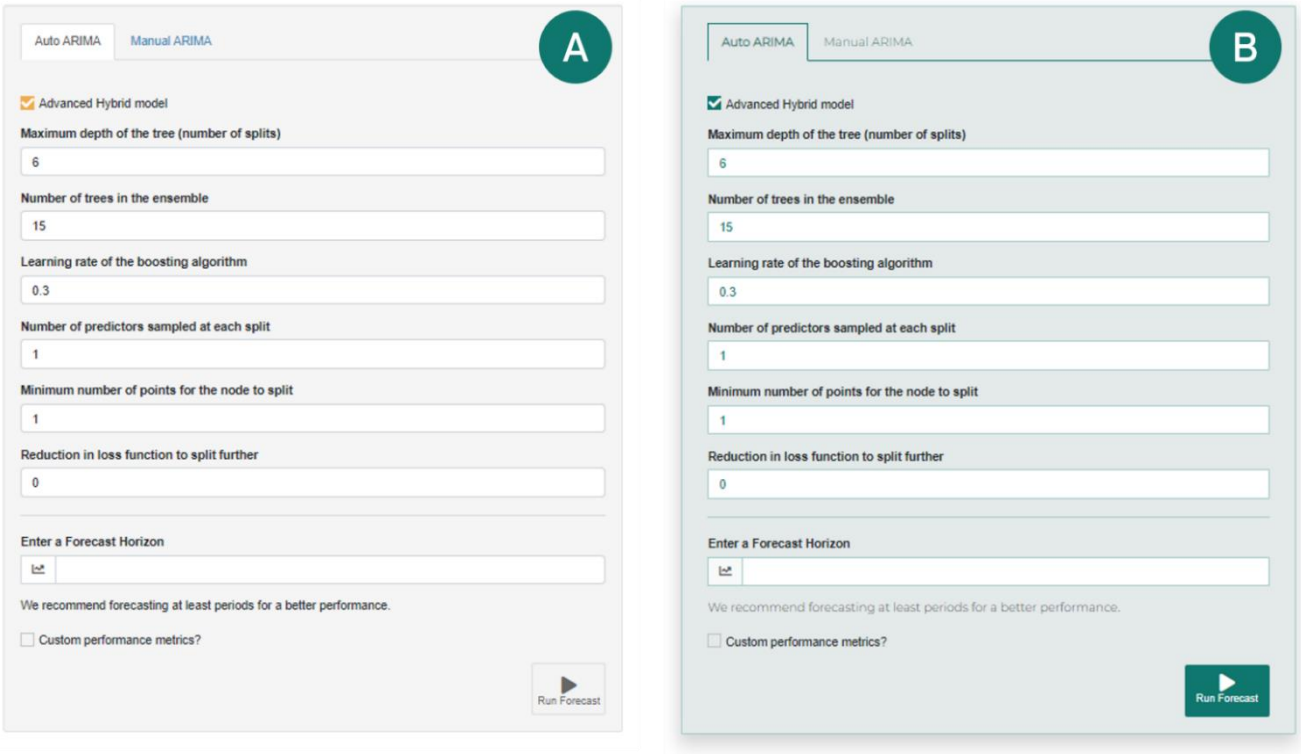
b. HTML

Además de los elementos que se pueden crear a través de funciones de R y que Shiny convierte en comandos de HTML, existe la posibilidad de incluir código puro de HTML si se conoce su sintaxis. Esta posibilidad permite generar una UI más customizada. El modo en que estos se incluyen en el *front-end* de la aplicación es a través de los *tags* de HTML y los más comunes ya están integrados como funciones de shiny. En el caso de Forecaster se han incluido *tags* como: `tags$a` para incluir hipervínculos en el texto, `tags$img` para incluir imágenes y `tags$b` para escribir texto en negrita.

c. Estilizado con CSS

Forecaster empezó teniendo un estilo muy marcado por los valores por defecto de los elementos creados con funciones del paquete shiny y paquetes equivalentes, como se puede apreciar en la Figura 10.A). La apariencia de muchos de estos elementos se ha modificado con minuciosidad para darle un aspecto diferenciado del que otras aplicaciones web Shiny suelen tener como se muestra en la Figura 10.B). Esto se ha logrado utilizando funciones de estilizado de CSS a través del método *inline* y se puede encontrar directamente en el código del objeto *ui* entre las líneas 190 y 596 del *script app.R*. El color que se ha escogido como tema principal de la aplicación es el RGB(16, 119, 111) ó #10776F en su valor Hexadecimal, y la fuente que se utiliza para todo el texto es Montserrat.

**Figura 10:** Estilizado de Forecaster por medio de CSS. **A)** aspecto de un wellpanel antes de la inclusión del código CSS. **B)** aspecto de un wellpanel después del estilizado con CSS.




*Fuente: elaboración propia a partir de impresiones de pantalla de Forecaster*

### *4.4.2. Server*

Después de definir los elementos visuales de la UI, se han de generar los *outputs* que se van a mostrar en esos elementos. Forecaster está desarrollada en su mayor parte utilizando R (R Core Team, 2021) como lenguaje de programación para generar dichos cálculos, análisis y motores; aunque también incluye *scripts* y funciones de Python (Van Rossum, Drake, 2018) para la sección de DL. Todo el código de R ha sido escrito utilizando RStudio como IDE (por las siglas en inglés de *Integrated Development Environment*) puesto que soporta todas las tecnologías que este proyecto utiliza.

a. R

Para el desarrollo de Forecaster se eligió R como lenguaje principal por diversos motivos. Desde un punto de vista funcional, el criterio clave para hacer la elección de R fue el medio en que se iba a querer mostrar el resultado del proyecto. Se escogió el *framework* Shiny porque resultaba muy adecuado para construir una aplicación web de esta magnitud y características sin necesidad de adentrarse en exceso en el mundo del desarrollo de software. Como se ha definido en el epígrafe 2.2., shiny es un paquete de código de R y, por tanto, la elección de R resulta evidente.

R es un lenguaje de programación muy presente en las participaciones de las notorias Competiciones Makridakis (Competiciones-M) de predicción de series temporales. Es tanto así que, en la Competición-M4 del año 2020 el 59% de los métodos que se presentaron utilizaron R como software para su desarrollo (Makridakis, Spiliotis y Assimakopoulos, 2020). Estas competiciones resultan de gran importancia para el avance en el estudio de metodologías para la predicción de series temporales debido a las conclusiones que de ella se obtienen. Es por eso que, el hecho de que las personas más innovadoras y conocedoras de la materia escojan este lenguaje para desarrollar sus contribuciones, genera cierta confianza en las posibilidades que R ofrece en este campo.

El número de paquetes disponibles en R a través de CRAN (*Comprehensive R Archive Network*) para trabajar con series temporales es muy elevado. Las *task view* de CRAN ofrecen una visión general de los paquetes más relevantes y disponibles para un tema de estudio específico y, para el caso del análisis de series temporales, hay disponibles en CRAN más de 300 paquetes (Anónimo, 2022). Escoger R por este motivo supone una gran ventaja, pues otros desarrolladores ya han creado funciones para virtualmente todas las tareas que uno pueda querer realizar con series de tiempo.

En relación con esta última idea, desde el principio del desarrollo de Forecaster se había escogido el paquete *modeltime* para conformar la base del modelado en que la aplicación se iba a asentar. Al ser *modeltime* un paquete desarrollado en R, existía un motivo más para su elección. Este paquete y los complementarios del ecosistema al que pertenecen se verán con más detenimiento en un epígrafe posterior.

Por último y no menos relevante, R también se ha elegido teniendo muy en cuenta un criterio personal y subjetivo. R es el lenguaje de programación con el que más experiencia

tengo y en el que más fluido soy a la hora de escribir código. Esta elección haría el proceso de desarrollo con R mucho más rápido, de provecho y consistente.

i. Paquetes utilizados

Forecaster ha requerido del uso y la integración de más de cincuenta paquetes distintos detallados en la Tabla 1. Todos ellos han sido instalados a través de CRAN a excepción de *neuralprophet* y *modeltime.gluonts*. Estos dos últimos han tenido que ser instalados directamente desde GitHub a través de la función `remotes::install_github()` debido a que, en el momento de su inclusión, todavía no habían sido publicados para su acceso desde CRAN.

**Tabla 1:** Paquetes de R referenciados en Forecaster

| Nombre del paquete | Versión | Acceso |
|---|---|---|
| tidyverse | 1.3.1 | CRAN |
| dplyr | 1.0.7 | CRAN |
| ggplot2 | 3.3.5 | CRAN |
| forcats | 0.5.1 | CRAN |
| readr | 2.0.1 | CRAN |
| tibble | 3.1.4 | CRAN |
| tidyr | 1.1.3 | CRAN |
| stringr | 1.4.0 | CRAN |
| DataExplorer | 0.8.2 | CRAN |
| berryFunctions | 1.20.1 | CRAN |
| tools | 4.1.1 | CRAN |
| reticulate | 1.20 | CRAN |
| bit64 | 4.0.5 | CRAN |
| remotes | 2.4.0 | CRAN |
| rlang | 0.4.11 | CRAN |
| ellipsis | 0.3.2 | CRAN |
| readxl | 1.3.1 | CRAN |

| | | |
|---|---|---|
| tidyquant | 1.0.3 | CRAN |
| data.table | 1.14.0 | CRAN |
| DT | 0.19 | CRAN |
| writexl | 1.4.0 | CRAN |
| fs | 1.5.0 | CRAN |
| rlist | 0.4.6.2 | CRAN |
| timetk | 2.6.1 | CRAN |
| lubridate | 1.7.10 | CRAN |
| imputeTS | 3.2 | CRAN |
| anytime | 0.3.9 | CRAN |
| smooth | 3.1.5 | CRAN |
| urca | 1.3.0 | CRAN |
| forecast | 8.15 | CRAN |
| fabble | 0.3.1 | CRAN |
| tidymodels | 0.1.3 | CRAN |
| modeltime | 1.0.0 | CRAN |
| modeltime.ensemble | 0.4.2 | CRAN |
| modeltime,resample | 0.2.0 | CRAN |
| modeltime.gluonts | 0.3.1 | GitHub: business-science/modeltime.gluonts |
| modeltime.h2o | 0.1.1 | CRAN |
| neuralprophet | 0.1.0 | GitHub: AlbertoAlmuinha/neuralprophet |
| TSrepr | 1.1.0 | CRAN |
| yardstick | 0.0.8 | CRAN |
| reactable | 0.2.3 | CRAN |
| plotly | 4.9.4.1 | CRAN |
| factoextra | 1.0.7 | CRAN |
| shiny | 1.6.0 | CRAN |
| shinydashboard | 0.7.1 | CRAN |
| shinycssloaders | 1.0.0 | CRAN |
| shinydashboardPlus | 2.0.2 | CRAN |
| shinyEffects | 0.2.0 | CRAN |

| | | |
|---|---|---|
| shinyWidgets | 0.6.1 | CRAN |
| shinyalert | 2.0.0 | CRAN |
| shinyjs | 2.0.0 | CRAN |
| fresh | 0.2.0 | CRAN |

*Fuente: elaboración propia a partir de CRAN (Anónimo, 2022)*

Debido al uso extenso que se ha hecho de algunos de los paquetes detallados en la tabla anterior, estos se verán con más detenimiento en un epígrafe a continuación.

b. Python

Por otra parte, se ha utilizado Python para la generación de los modelos de predicción de *Deep Learning*, en concreto la especificación del algoritmo DeepAR. Este código de Pyhton se ha escrito en un *script* a parte utilizando la librería *GluonTS* y se ha integrado con el resto de los motores de cálculo del *server* a través del paquete *reticulate* de R.

i. *reticulate*

Este paquete ofrece un amplio elenco de herramientas para poder integrar la funcionalidad de Python dentro de un programa de R y viceversa. *Reticulate* permite ejecutar comandos y *scripts* enteros de Python desde R. Además, posibilita la configuración de entornos virtuales y entornos de Conda para poder utilizar la versión de Python que las librerías en uso requieren (Ushey, Tang y Allaire, 2022).

Otro de los componentes de Forecaster que requirió el uso de *reticulate* fue el paquete *modeltime.gluonts* debido a que las funciones que tiene definidas utilizan Python por detrás. De este modo, un entorno virtual con la versión 3.7.1 de Python instalado en él tuvo que ser configurado desde R y su contenido es el siguiente:

**Tabla 2:** Dependencias de Python en el entorno virtual

| Nombre de la librería | Versión | Acceso |
|---|---|---|
| mxnet | 1.7 | pip |
| gluonts | 0.8.0 | pip |
| numpy | *NA* | pip |
| pandas | 1.0.5 | pip |
| pathlib | 1.0.1 | pip |
| ujson | 4.0.2 | pip |
| brotlipy | 0.7.0 | pip |
| torch | 1.6 | pip |
| pytorch-lightning | 1.1 | pip |

*Fuente: elaboración propia a partir de (Dancho, n.d.)*

### *4.4.3. Despliegue y puesta en producción*

Una vez se ha escrito toda la aplicación Shiny esta se ha de poder mostrar a todos aquellos usuarios para quien se ha diseñado y pensado. Como se mencionó en el epígrafe 2.1. sobre el modo de acceso a una aplicación web, este puede hacerse de modo local utilizando la dirección *localhost* del ordenador del usuario o a través de una conexión a un servidor. Forecaster permite su uso a través de ambas vías.

a. Despliegue local

Para que el usuario pueda hacer uso de Forecaster a través de su propio ordenador, este ha de descargar todo el código que define la aplicación. Además, ha de tener los programas necesarios para su ejecución, en este caso deberá contar como mínimo con R y Shiny instalados en su ordenador. Como se verá a continuación, el usuario también deberá realizar ciertas configuraciones adicionales para algunas de las funcionalidades más avanzadas que Forecaster ofrece.

Durante el proceso de desarrollo de Forecaster se ha utilizado Git como software para el control de versiones de todo el código. Esta herramienta es una pieza fundamental en el

proceso de desarrollo pues permite mantener una trazabilidad exacta de todos los cambios realizados en el proyecto. Git se basa en los repositorios o "repos" como lugar central donde se guarda todo el código y demás materiales con los que se trabaja.

El código de la aplicación está almacenado en un repo público de GitHub que es accesible a través de la siguiente dirección *url*: github.com/pablochaure/Forecaster. GitHub es un servicio web basado en la nube que funciona como un repositorio remoto y utiliza Git como tecnología fundacional. Se trata de una "copia" del contenido existente en el repo local desde el que realmente se trabaja. El uso de GitHub ayuda a los desarrolladores a almacenar y gestionar su código, a publicarlo y compartirlo con otros desarrolladores, así como a hacer un seguimiento y controlar los cambios en el mismo.

Conociendo estos detalles acerca de la ubicación de los archivos de Forecaster, el usuario deberá instalar Git en su ordenador para poder clonar directamente el repo remoto en su directorio local. Esta opción es la preferida por la sencillez que presenta y porque se asegura que la copia es exacta a la que existe en el repo remoto. Alternativamente, podría descargar el contenido del repo de forma manual y descomprimir su contenido en el directorio deseado.

Una vez dispone del repositorio en su ordenador no tiene más que abrir el archivo *app.R* o el proyecto *Forecaster.Rproj* y correr la aplicación directamente. Esto lo podrá hacer, bien ejecutando el comando `shinyApp(ui, server)` ubicado en la línea 4464 de *app.R* o bien, en el editor de RStudio, pulsando el botón de *Run App* de la esquina superior derecha de la pantalla *source*.

Existen dos particularidades en la instalación local para algunas funcionalidades más avanzadas. Para poder hacer uso de la automatización de entrenamiento de modelos de ML (AutoML) de Forecaster, el usuario tiene que disponer de una instalación de Java en una versión superior a la ocho (Dancho, 2021). Este es un requerimiento del sistema especial del paquete *modeltime.h2o* y que se hereda de la configuración del software H2O.ai. Por otra parte, para poder entrenar alguno de los modelos de DL disponibles, el usuario ha de configurar el entorno virtual de Python en su sesión de RStudio con las dependencias que se han detallado en el epígrafe anterior.

b. Despliegue en servidor

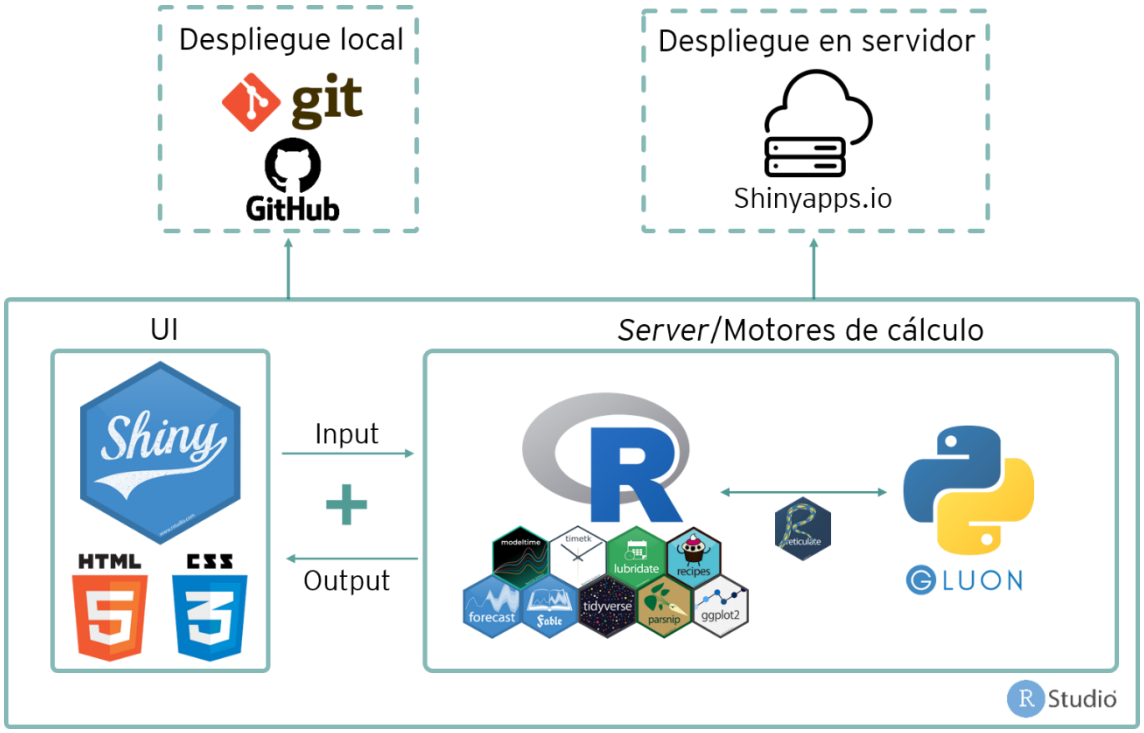
El usuario también es capaz de acceder a Forecaster a través de la dirección url: pablochaure.shinyapps.io/Forecaster desde cualquier navegador de Internet. Para posibilitar el acceso a través de una red de Internet, se ha utilizado el software Shiny Server de RStudio. En nuestro caso, se ha optado por utilizar los servidores que RStudio pone a disposición del público a través de su servicio Shinyapps.io.

Shinyapps.io se trata de una plataforma como servicio o PaaS (por las siglas en inglés de *Platform as a Service)* fácil de configurar y en el que desplegar aplicaciones, incluye encriptación del contenido y es fácilmente escalable para satisfacer la carga de trabajo demandada. La mayor ventaja que ofrece es evitar al desarrollador que tenga que configurar y mantener su propio servidor. No obstante, el que esté alojada en servidores remotos implica que todo el código y los datos necesarios han de ser copiados en él por estar ofrecido el servicio por RStudio y se pierde cierta privacidad del código.

El despliegue de la aplicación se puede hacer directamente desde R a través de las funciones del paquete de R *rsconnect*. Este paquete detecta todos los paquetes que la aplicación utiliza, los compila y los manda al servicio de Shinyapps.io. Por su parte, Shinyapps.io instala los paquetes en el entorno del servidor de la aplicación la primera vez que se despliega y se actualizan en las sucesivas actualizaciones que se hagan del contenido.

El servidor en el que está alojada Forecaster es de la categoría *large* y tiene una memoria de 1 GB. Con su contenido y paquetes actuales no es necesario aumentar la categoría del servicio, aunque es muy probable que con la inclusión de mayores funcionalidades esto se deba reconsiderar en el futuro.

**Figura 11:** Esquema global de la arquitectura de Forecaster




*Fuente: elaboración propia*

## 4.5. Motores de cálculo

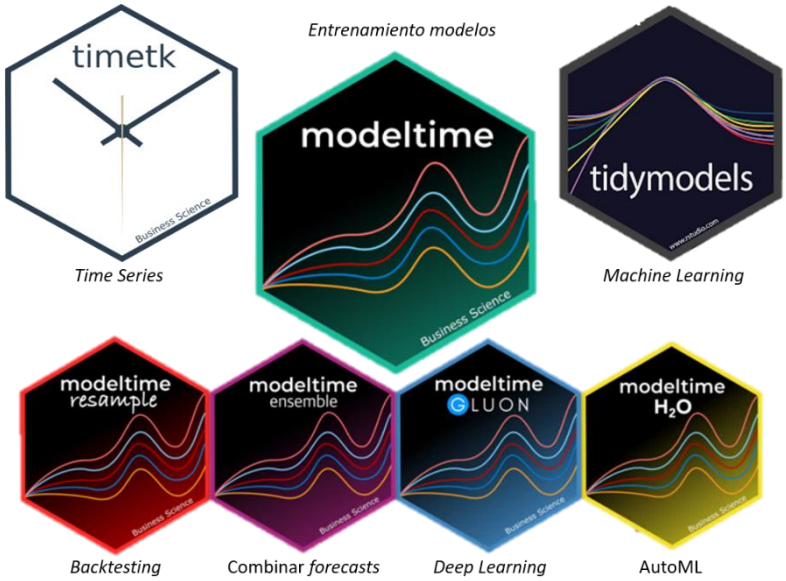
Como se ha detallado en la Tabla 1, Forecaster está diseñada con una amplia variedad de paquetes de los cuales alrededor del 70% son utilizados para tareas de análisis, generación de modelos y visualizaciones. De ellos, los paquetes del ecosistema *modeltime* son la pieza fundamental del funcionamiento y por ello se detallan algunas de sus características y particularidades. Por otra parte, se han definido numerosas funciones propias para la adaptación de las funciones de los paquetes al diseño de Forecaster y para extender su aplicación al caso de uso deseado.

### *4.5.1. Ecosistema Modeltime*

*Modeltime* es un ecosistema diseñado para albergar todas las herramientas necesarias para analizar series temporales y entrenar modelos de ML, DL y modelos clásicos o

estadísticos. Es una extensión del conocido paquete *tidymodels* y sus componentes para realizar *forecasting* con el estilo *tidy* propio de R. Está concebido para acelerar la experimentación de modelos a través de su característico *workflow* que permite entrenar, evaluar, seleccionar y proyectar modelos fácilmente. Este *workflow* se verá en detalle en un epígrafe posterior.

**Figura 12:** Paquetes que conforman el ecosistema *modeltime*




*Fuente: elaboración propia*

El ecosistema está formado por cinco paquetes para el modelado que heredan el diseño de *tidymodels* y, además, trabaja en conjunción con *timetk* para todo lo relativo al tratamiento de series temporales como sets de datos. No se ofrecen detalles de *modeltime.resample* debido a que no está integrado en la aplicación por el momento.

a. Modeltime

*Modeltime* es la pieza central del ecosistema y donde quedan definidas las funciones esenciales para todo el modelado. Sirve como base para integrar las capacidades del resto de paquetes del ecosistema en un mismo modo de trabajo definiendo una sintaxis común a todos ellos.

Sus desarrolladores lo diseñaron para agilizar el proceso de generación de modelos y la comparación entre ellos y, de este modo, evitar la repetición en el entrenamiento. Esto es posible gracias a la integración del ecosistema de paquetes de ML *tidymodels* en un flujo de trabajo propio (Dancho, 2020).

*Modeltime* es un paquete que integra las posibilidades de modelos de ML, modelos clásicos y modelos automatizados para la predicción de series temporales. Los modelos de ML provienen del paquete *parsnip* que incluye una selección de más de 40 modelos de distintos motores o fuentes como son *randomforest* y *kernlab*, por mencionar alguno (Kuhn, n.d.). Por otra parte, los modelos clásicos y automáticos son modelos propios de *modeltime* pero que están tomados del famoso paquete de R *forecast* y de la librería *prophet* desarrollada por Facebook.

b. Timetk

*Timetk* es un paquete para el manejo de datos de series de tiempo y abarca las tareas de visualización de datos, limpieza de datos, preprocesamiento y modificación y creación de variables. Muchas de las funcionalidades de este paquete provienen de otros ya en uso pero que *timetk* les da un nuevo enfoque para integrarlas todas en un mismo lugar. Por ejemplo, las funciones que utilizan la frecuencia de la serie como argumento dependen del paquete *stats*; la periodicidad y la automatización de *lags* se basan en funciones de *xts* (Dancho, n.d.). La funcionalidad que más ha aprovechado Forecaster de este paquete es todo lo relativo a la visualización de datos interactiva y estática que ofrecen funciones como `plot_timeseries()`, `plot_stl_diagnostics()`, `plot_acf_diagnostics()` y `plot_seasonal_diagnostics()`.

c. Modeltime H2O

El paquete *modeltime.h2o* está dedicado a la generación automatizada de modelos de ML (AutoML) para la predicción de series temporales. Este paquete integra las posibilidades de H2O AutoML, una plataforma *open source* para la generación rápida, fácil y escalable de modelos de ML desarrollada por H2O.ai.

Este paquete se ha utilizado para desarrollar la sección de AutoML de Forecaster integrándolo en el flujo de trabajo del resto de la aplicación a través de la función `modeltime.h2o::automl_reg()`. Esta función entrena y valida con *cross-validation* múltiples modelos de ML y DL (XGBoost GBM, GLMs, Random Forest, GBMs...). Además, entrena dos modelos *Stacked Ensemble*, uno de todos los modelos, y otro sólo de los mejores modelos de cada tipo o "familia". Por último, se selecciona el mejor modelo en función de una métrica de precisión predefinida por el usuario (Dancho, n.d.).

d. Modeltime Ensemble

Este paquete integra técnicas de ensamblado de modelos a través de tres estrategias: *stacked ensembling* utilizando un modelo de regresión *Elastic Net*, ensamblado ponderado y ensamblado promedio. Los modelos de Forecaster que se pueden ensamblar con este paquete actualmente corresponden a los disponibles en la sección de ML.

e. Modeltime GluonTS

El último paquete del ecosistema es también el más avanzado de todos ellos. *Modeltime.gluonts* integra la librería GluonTS de Python para el entrenamiento de modelos de DL. Esta es una colección de modelos y técnicas desarrollada por Amazon Web Services para todas las necesidades de *forecasting* con DL. Esta integración hace fácil el entrenamiento de esta clase de modelos y la posibilidad de incluirlos dentro del *workflow* de *modeltime*. Actualmente su mantenimiento se lleva a cabo a través de GitHub y los últimos desarrollos no están disponibles a través de CRAN (Dancho, n.d.).

Las funciones que ofrece este paquete provienen de Python, por lo tanto, es necesaria la instalación de un entorno virtual con las dependencias correspondientes como se detalla en el apartado de la arquitectura para Python. Esta instalación está facilitada por la función interna del paquete `modeltime.gluonts::install_gluonts()` y permite la conexión con el paquete de Python.

### *4.5.2. Funciones propias*

Además de todas las funciones de los paquetes disponibles a través de CRAN, Forecaster ha requerido, para las funcionalidades específicas que se querían ofrecer, la escritura y declaración de funciones propias. Estas funciones han sido necesarias para diseñar desde cero características que la aplicación debía tener y que no se encontraban o se desconocían de otros paquetes publicados. Pero no solo se han creado funciones integralmente, también ha sido necesaria la manipulación y adaptación de algunas de otros paquetes. Para hacer esto, se ha obtenido a través de GitHub el código fuente que las define y se ha identificado la modificación a realizar para lograr el resultado deseado.

En la tabla a continuación se detallan las más de treinta funciones declaradas para Forecaster junto con una breve descripción de su finalidad.

**Tabla 3:** Funciones propias definidas para Forecaster

| Nombre de la función | Descripción |
|---|---|
| libraries() | Carga de todos los paquetes necesarios. Si no están instalados, se instalan y se cargan |
| frequency_data() | Obtención de la frecuencia de la variable de fecha |
| sliderInput2() | Modificación de los atributos del sliderInput() de *shiny* |
| read_data() | Carga de los archivos CSV con distintos formatos |
| select_data() | Creación del *dataframe* a partir de las variables seleccionadas |
| count_rangeselector() | Ajuste del *selector* de plotly para cada frecuencia |
| plot_acf | Modificación de timetk::plot_acf_diagnostics() para adecuarlo a la reactividad |
| accuracy_metrics() | Recopilación de todas las métricas seleccionadas |
| mse() | Cálculo del MSE |
| mae() | Cálculo del MAE |
| mape() | Cálculo del MAPE |
| mase() | Cálculo del MASE |
| smape() | Cálculo del SMAPE |
| rmse() | Cálculo del RMSE |

| | |
|---|---|
| maape() | Cálculo del MAAPE |
| manual_arima() | Entrenamiento del modelo ARIMA con parámetros seleccionados manualmente |
| auto_arima() | Entrenamiento del modelo ARIMA automatizado |
| ses_forecast() | Entrenamiento del modelo ETS simple |
| holt_forecast() | Entrenamiento del modelo ETS doble o de Holt |
| holtwinters_forecast() | Entrenamiento del modelo ETS Holt-Winters' |
| ets_models() | Recopilación de todos los modelos de ETS |
| ets_custom() | Modificación de forecast::ets() para incluir los modelos ETS manuales |
| exp_smoothing_custom() | Modificación de modeltime::exp_smoothing() para incluir los modelos ETS manuales |
| prophet_forecast() | Entrenamiento del modelo Prophet |
| prophetboost_forecast() | Entrenamiento del modelo ProphetBoost |
| randomforest_forecast() | Entrenamiento del modelo Random Forest |
| xgboost_forecast() | Entrenamiento del modelo XGBoost |
| ml_models() | Recopilación de todos los modelos de Machine Learning |
| get_loadings() | Obtención de los pesos de cada modelo para el ensamblado con media ponderada |
| ensemble_calibration() | Entrenamiento de los modelos de ensamblado |
| frequency_to_pandas() | Transformación de la frecuencia al formato de Pandas |
| deepar_forecast() | Entrenamiento del modelo DeepAR |
| nbeats_forecast() | Entrenamiento del modelo N-Beats Ensemble |
| DL_models() | Recopilación de todos los modelos de DL |

*Fuente: elaboración propia*

### 4.6. Estructura de Forecaster

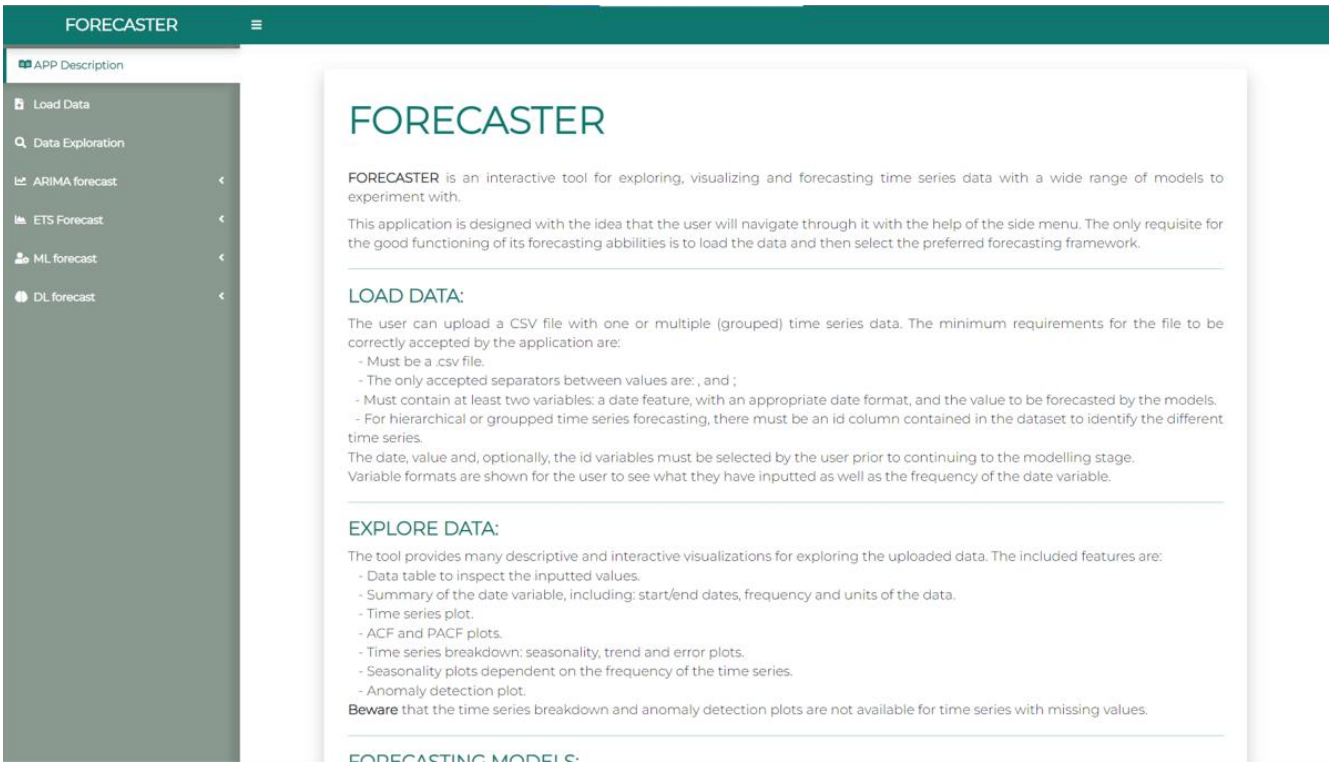
El objetivo de este epígrafe es proporcionar una descripción del proceso de funcionamiento del software, sugiriendo una secuencia de pasos que el usuario suele

seguir cuando explora la aplicación. Esto tiene el objetivo de proporcionar directrices y servir como un manual de usuario básico. El flujo de trabajo de Forecaster se definió de tal modo que imite, en parte, el proceso natural propio de la ciencia de los datos. Este flujo se compone de cinco fases:

1. Carga de los datos
2. Análisis exploratorio y visualización de los datos introducidos
- Modelado:
    3. Selección del modelo/s a entrenar
    4. Entrenamiento y validación del modelo/s
    5. Predicción hacia el futuro

La primera ventana que aparece siempre que se inicia Forecaster corresponde con una página de bienvenida en la que se describe la estructura y el funcionamiento de la aplicación a grandes rasgos. Esta página tiene contenido esencial que entender para poder aprovechar todo el potencial que ofrece Forecaster.

**Figura 13:** Página de bienvenida y descripción de Forecaster




*Fuente: elaboración propia a partir de una impresión de pantalla de Forecaster*

### *4.6.1. Carga de los datos.*

Después de la pestaña de bienvenida con la descripción de la app, el usuario debe acceder a la carga de los datos (*Load Data*). Esta pestaña cuenta con distintos componentes en una misma ventana.

En primer lugar, al principio de la página se encuentra una caja con un *fileinput* (Figura 15.A) para la introducción del archivo que contiene los datos que se pretenden analizar y predecir. El usuario puede seleccionar de su directorio local un archivo con una o múltiples series temporales. Han de cumplirse ciertos requisitos para que la carga se realice satisfactoriamente. Primero, el archivo debe estar en formato CSV. Segundo, los valores del archivo han de estar separados por coma (formato estándar CSV) o por punto y coma. Tercero, debe contener como mínimo dos columnas que se categorizarán después: una variable de fecha con un formato de fecha apropiado y una variable objetivo que es lo que se va a querer proyectar. Por último, en caso de que se introduzca un archivo con múltiples series temporales, debe existir una tercera columna que sirva de identificador para cada una de las series. Idealmente, la tabla de entrada tiene un aspecto similar al siguiente:

**Figura 14:** Formato ideal de la tabla de datos cargada. **A)** Tabla con una única serie temporal. **B)** Tabla con identificador para múltiples series temporales

| Fecha | Valor |
|---|---|
| 2021-01-01 | 383 |
| 2021-01-02 | 412 |
| 2021-01-03 | 420 |
| 2021-01-04 | 395 |
| 2021-01-05 | 348 |
| 2021-01-06 | 387 |
| 2021-01-07 | 392 |
| … | … |

A

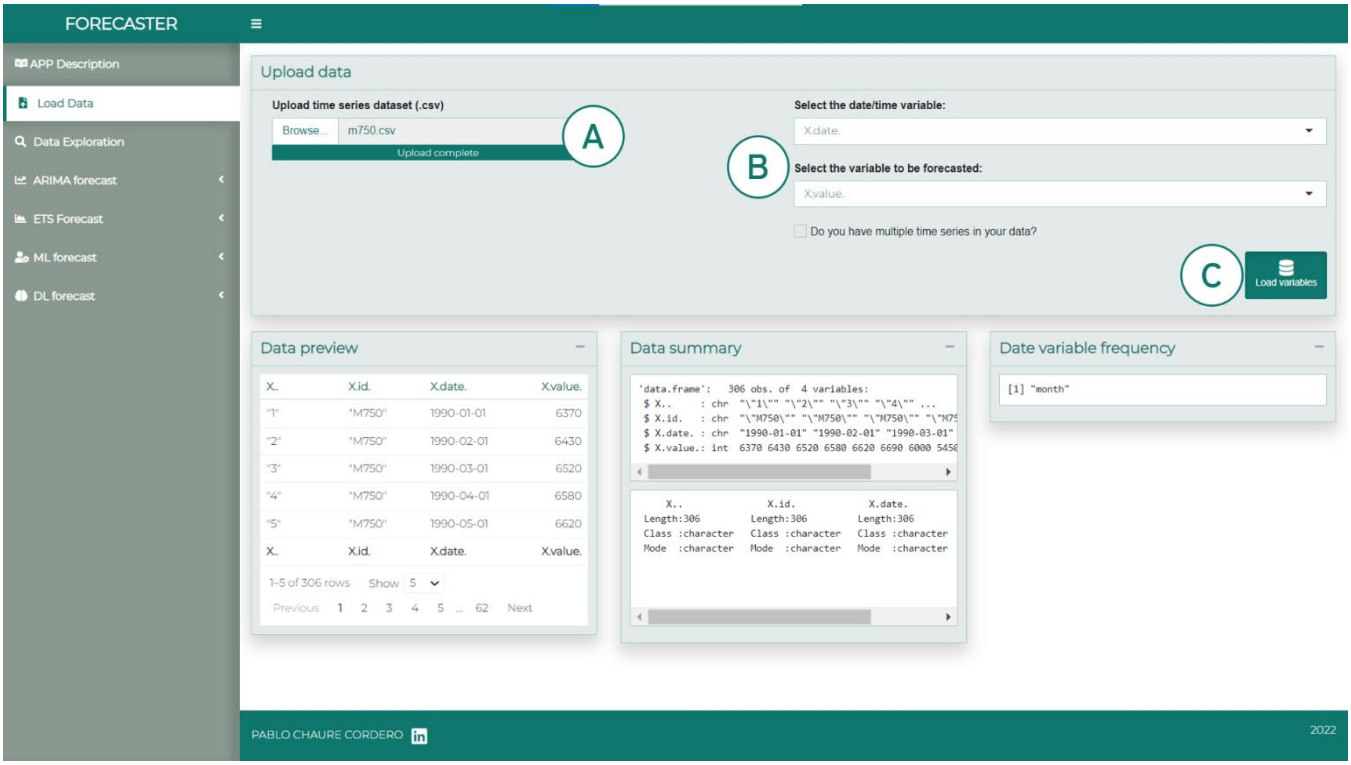
| Id | Fecha | Valor |
|---|---|---|
| 01_05 | 2021-01-01 | 383 |
| 01_05 | 2021-01-02 | 412 |
| 01_05 | 2021-01-03 | 420 |
| … | … | … |
| 01_09 | 2021-01-01 | 237 |
| 01_09 | 2021-01-02 | 341 |
| 01_09 | 2021-01-03 | 298 |
| … | … | … |

B

*Fuente: elaboración propia*

A la derecha de este *fileinput*, hay distintas listas desplegables (Figura 15.B) para la categorización de las variables cargadas. El usuario debe, como mínimo, seleccionar la variable correspondiente a la fecha y la variable a predecir del archivo cargado que se muestran en los desplegables. En el caso de existir series temporales agrupadas en el archivo, se ha de marcar la caja tal y como se indica y seleccionar la variable con el identificador. Una vez se ha realizado esta clasificación basta con pulsar el botón de *Load Variables* (Figura 15.C) para que queden los datos cargados y no haya que encargarse de ello más.

**Figura 15:** Pantalla de carga de los datos. **A)** *Fileinput.* **B)** Listas desplegables. **C)** Botón de carga de variables.




*Fuente: elaboración propia a partir de una impresión de pantalla de Forecaster*

Por último, se muestran otras tres cajas en la parte inferior de la pantalla con una previsualización de los datos cargados, un resumen y detalle breve de todas las variables existente en los datos; y una referencia a la frecuencia temporal en que está la variable de fecha.

### *4.6.2. Análisis exploratorio y visualización de los datos introducidos*

En la siguiente pestaña del menú lateral o *sidebar* se ubica la segunda parte del flujo de Forecaster correspondiente al análisis exploratorio y la visualización de la serie temporal. Esta pestaña, a diferencia de la anterior, está dividida en seis subpestañas que se describen de izquierda a derecha. La totalidad de los *outputs* que se han generado provienen de funciones del paquete *timetk*.

En primer lugar, se muestra un breve resumen de la variable temporal que describe el número de observaciones de la serie, la fecha de inicio y fin; la unidad temporal o granularidad en que están las fechas y la frecuencia de esta. Este resumen proviene de la función `timetk::tk_summary_diagnostics()`.

A continuación, se presenta el gráfico de línea de la serie temporal con ciertos parámetros de la visualización que se pueden ajustar. Es posible aplicar una transformación logarítmica para reducir la escala y añadir o quitar una media móvil, junto con el grado de suavizado de esta que se desea mostrar.

**Figura 16:** Visualización de la serie temporal cargada


*Fuente: elaboración propia a partir de una impresión de pantalla de Forecaster*

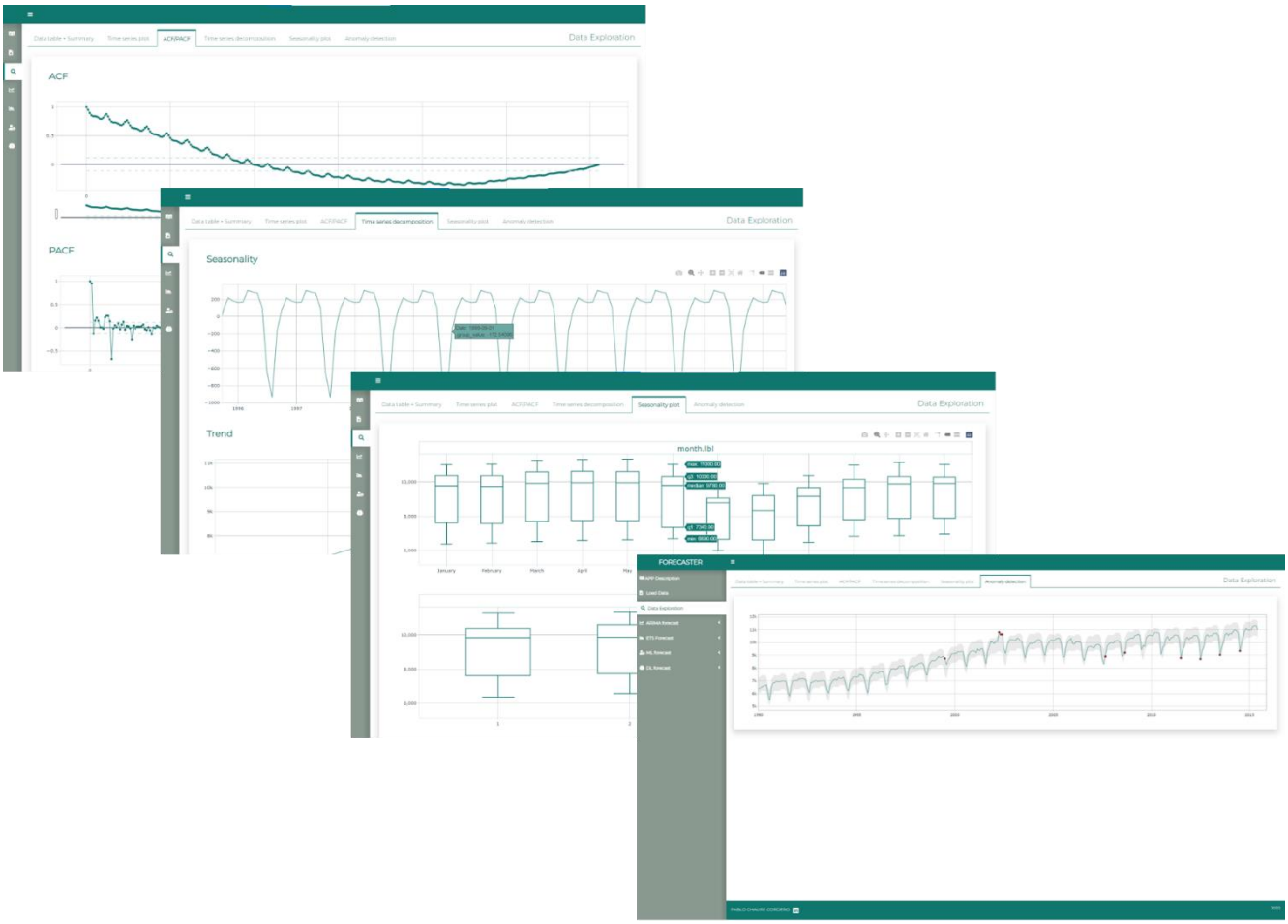
La siguiente pestaña corresponde a las visualizaciones de las funciones de autocorrelación y de autocorrelación parcial. Es el *output* de la función `timetk::plot_acf_diagnostics()` que se adaptó en la función `f_plot_acf()` para poder mostrar cada uno de los gráficos con su respectivo título y sección en lugar de obtener las dos con una misma llamada. Estos gráficos incluyen intervalos de significación calculados con $\pm 2/sqrt(T)$ siendo T la longitud de la serie temporal (Dancho, n.d.).

En la pestaña de *Time series decomposition* se muestran tres gráficos interactivos que corresponden a la estacionalidad, la tendencia y el error que, conjuntamente, conforman la serie cargada. Esto es de especial utilidad para entender los componentes básicos de la serie para modelos como los de suavizado exponencial. Se ha generado con la función `timetk::plot_stl_diagnostics()`.

Seguidamente, se visualizan los gráficos de estacionalidad de la serie. El número de gráficos que se muestren dependerá de la frecuencia en que estén los datos. Para poder lograr esto se ha diseñado de tal modo que la frecuencia sea un parámetro para una función anónima que a su vez llama a `timetk::plot_seasonal_diagnostics()`. De este modo si, por ejemplo, la serie tiene frecuencia mensual se calcularán los *boxplots* para la estacionalidad mensual, trimestral y anual.

Por último, se pueden analizar los *outliers* o puntos anómalos que presenta la serie en la pestaña *Anomaly detection*. Se utiliza la función `timetk::plot_anomaly_diagnostics()` que realiza una descomposición de la serie y sustrae la tendencia y la estacionalidad para detectar los *outliers* en el resto de la serie (Dancho, n.d.).

**Figura 17:** Análisis exploratorio




*Fuente: elaboración propia a partir de impresiones de pantalla de Forecaster*

### *4.6.3. Modelado de series temporales*

Las últimas tres partes del proceso de Forecaster versan en torno al entrenamiento y proyección de los modelos de predicción de la serie temporal cargada. Como se ha mencionado en la introducción de este capítulo, Forecaster cuenta con cuatro categorías de modelos que atienden a distintas tipologías: ARIMA, ETS, ML y DL. Para todas ellas, Forecaster tiene diseñado un flujo de trabajo estándar que se basa en gran parte en el ya mencionado *workflow* de *modeltime*. Además de esto, existen ciertas características y funcionalidades comunes a todas las secciones de modelos como el cálculo de las métricas de precisión, la definición de los puntos predichos y sus intervalos de confianza; y el diseño de la UI.

a. Flujo de trabajo para el modelado

i. Extensión de los datos hasta el horizonte futuro

El primer paso del flujo de trabajo consiste en extender el set de datos para incluir la ventana de tiempo futuro en que se quiere proyectar el valor. Para esto se requiere de la especificación de ese horizonte por parte del usuario a través del *input* para ello. Forecaster computa y muestra por defecto un horizonte sugerido que coincide con un 18% de todas las observaciones de la serie introducida para que haya un *split* balanceado. No obstante, el usuario puede sobrescribirlo para adaptarlo a su caso. La extensión de los datos se hace a partir de la función `timetk::future_frame()`.

ii. División o s*plit* de los datos en set de entrenamiento y de test

Antes de obtener los sets de entrenamiento y test, el *dataframe* resultante de la extensión anterior se divide en los datos actuales o cargados y el set futuro. Tomando los datos actuales, se realiza el *split* para obtener el set con el que se va a entrenar el modelo y el set de test con el cual se mide la precisión y la capacidad de predicción del modelo. Esta división se realiza tomando el mismo número de periodos que el horizonte para determinar el set de test. De este modo, el valor recomendado por defecto del 18% puede ser aumentado o disminuido por el usuario a su antojo, pero debe tener en cuenta que puede incurrir en dos sets muy desequilibrados y que el modelo no logre captar las señales más cercanas al momento en que se quiere predecir. El *split* se realiza a través de la función `timetk::time_series_split()`.

iii. Selección y entrenamiento de los modelos

El siguiente paso consiste en la selección del modelo/s que se quiere entrenar con las características que desea especificar el usuario. Como se ha mencionado en varias ocasiones, existen algunos modelos automatizados y otros que permiten la definición de

los parámetros específicos si el usuario es más experimentado en la materia. Este sería el momento para precisarlos.

Con estos detalles, el/los modelos/s se entrenarían con los datos del set de entrenamiento. Se ha de añadir que, en el proceso de entrenamiento y específicamente para los modelos de ML y DL, la fórmula del modelo que se entrena contiene variables independientes adicionales además de la fecha. De este modo, se incluye un preprocesador que descompone la fecha automáticamente en variables más informativas como puede ser el día de la semana, el día del año o el trimestre. Esto es posible gracias a la función `timetk::tk_augment_timeseries_signature()` que genera más de 25 *features* diferentes de las cuales se descartan algunas por carecer de relevancia.

iv. Creación de la tabla de modelos

A partir de este cuarto paso comienza el *workflow* de *modeltime* que se lleva a cabo internamente en el funcionamiento de la aplicación. Se inicia con la creación de una tabla de modelos (`modeltime_table()`) y sigue con la inclusión de todos los modelos especificados. Este paso realiza algunas comprobaciones básicas para asegurarse de que cada uno de los modelos incluidos se ha entrenado adecuadamente y que se organizan en una estructura escalable. La *modeltime table* sirve como base para el resto de los pasos del *workflow*.

v. Calibración y predicción de los modelos en el set de test

La calibración consiste en añadir la columna `.calibration_data` a la tabla anterior con las predicciones en el test set y los residuos entre el valor real y el predicho. Este paso es en el cual se computan y determinan los intervalos de confianza de las predicciones, así como las métricas de precisión. Después de la calibración, el *workflow* continúa con estos nuevos datos añadidos.

vi. Cálculo de las métricas de precisión de los modelos

Para evaluar el modelo entrenado y su utilidad para los datos dados, se puede hacer a partir de un análisis visual del *forecast* en comparación con el set de test y tomando como referencia ciertas métricas de precisión del modelo. Forecaster utiliza la función `modeltime_accuracy()` para calcular estas medidas y se muestran al usuario en una tabla interactiva gracias a `table_modeltime_accuracy()` que permite la comparativa de modelos en términos de su capacidad de predicción en el set de test.

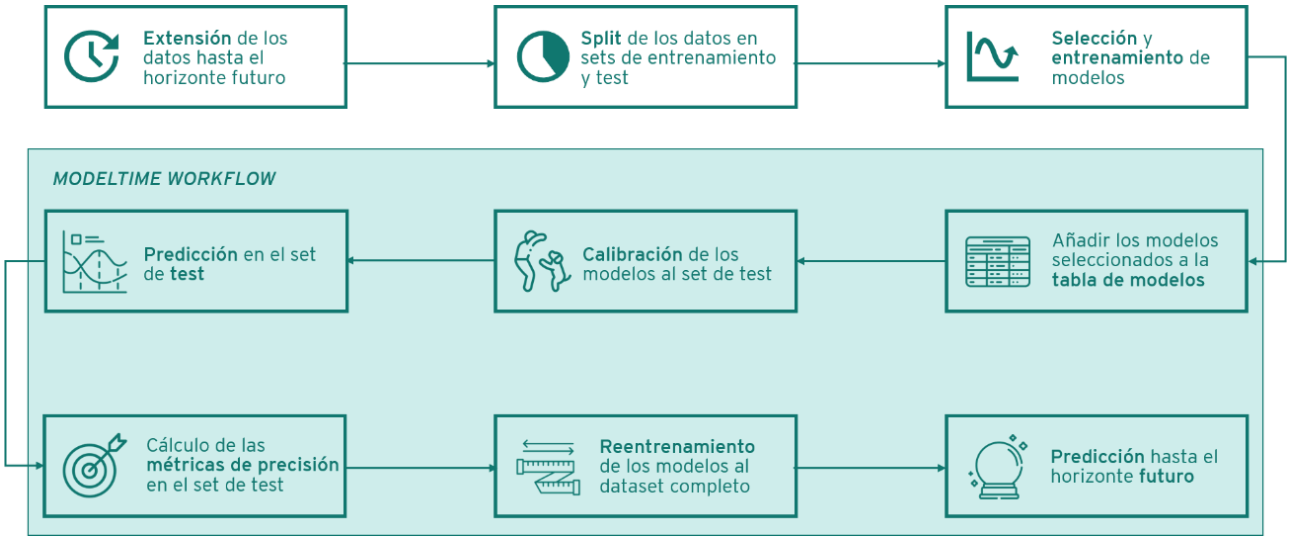
vii. Reentrenamiento de los modelos al set de datos completo

Una vez se han evaluado los modelos entrenados, el objetivo es proyectar la serie temporal hacia el futuro. Para obtener mejores resultados en este paso, lo ideal es utilizar toda la información disponible para obtener las señales más explicativas del futuro. En una serie temporal estas señales se encuentran en los periodos más cercanos al momento en que se quiere proyectar y es por eso que es de gran valor entrenar de nuevo el modelo con toda la serie de datos (set de entrenamiento y de test, conjuntamente). En este punto puede ser que, para los modelos automatizados, la información más reciente suponga una modificación de los parámetros seleccionados y el modelo con el que se va a proyectar la serie cambie del que se había evaluado previamente.

viii. Predicción hasta el horizonte futuro

Utilizando el modelo actualizado en el paso anterior y que capta las señales más recientes de la serie, se trata ahora de cumplir el objetivo principal de la aplicación que es predecir los valores futuros de la serie. Para ello se sirve del horizonte introducido en el primer paso del flujo y hasta el cual la serie va a quedar proyectada. Este es el último paso del proceso y en el que se obtienen los valores proyectados de la mediana, así como un intervalo de confianza del 95% para los mismos.

**Figura 18:** Flujo de trabajo del modelado




*Fuente: elaboración propia*

b. Métricas de precisión

En el sexto paso del flujo de trabajo detallado anteriormente se menciona el cálculo de distintas métricas para evaluar el modelo entrenado con datos fuera de la muestra. Forecaster ofrece al usuario la posibilidad de escoger entre una selección de nueve métricas diferentes cuáles quiere que se calculen para evaluar los modelos. Existe un conjunto de seis métricas seleccionado por defecto que se puede ampliar con hasta esas tres más si el usuario así lo desea y estima de valor. Las nueve métricas quedan descritas en la Tabla 4.

La elección de las métricas a computar requiere de conocimientos avanzados en la materia y no se espera que el usuario más funcional entienda el valor específico de cada una de ellas. No obstante, para el usuario más técnico esta posibilidad aporta gran valor analítico y flexibilidad para la evaluación de los modelos.

Las métricas incluidas provienen en su totalidad del paquete *yardstick*, a excepción de MSE, MPE y MAAPE que han sido incluidas a través de funciones propias de Forecaster por no estar disponibles en dicho paquete.

**Tabla 4:** Métricas de precisión disponibles en Forecaster[6]

| Métrica | Nombre completo | Cálculo[7] | Conjunto | Objetivo |
|---|---|---|---|---|
| MAE | Mean Absolute Error | $\frac{1}{n}\sum_{t=1}^{n}\lvert y_i - x_i\rvert$ | Por defecto | Minimizar |
| MAPE | Mean Absolute Percentage Error | $\frac{100\%}{n}\sum_{i=1}^{n}\left\lvert\frac{x_t - y_t}{x_t}\right\rvert$ | Por defecto | Minimizar |
| MASE | Mean Absolute Scaled Error | $\frac{\frac{1}{j}\sum_j \lvert e_j\rvert}{\frac{1}{T-1}\sum_{t=2}^{T}\lvert y_t - y_{t-1}\rvert}$ | Por defecto | Minimizar |
| SMAPE | Symmetric Mean Absolute Percentage Error | $\frac{100\%}{n}\sum_{i=1}^{n}\frac{\lvert y_t - x_t\rvert}{(\lvert x_t\rvert + \lvert y_t\rvert)/2}$ | Por defecto | Minimizar |
| RMSE | Root Mean Squared Error | $\sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{n}}$ | Por defecto | Minimizar |
| $R^2$ | R Squared | $1 - \frac{SSR}{SST} = 1 - \frac{\sum(\hat{y}_t - \bar{y})^2}{\sum(y_t - \bar{y})^2}$ | Por defecto | Maximizar |
| MSE | Mean Squared Error | $\frac{1}{n}\sum_{t=1}^{T}(\hat{y}_t - y_t)^2$ | Adicional | Minimizar |
| MPE | Mean Percentage Error | $\frac{100\%}{n}\sum_{i=1}^{n}\frac{x_t - y_t}{x_t}$ | Adicional | Minimizar |
| MAAPE | Mean Arctangent Absolute Percentage Error | $\frac{1}{n}\sum_{i=1}^{n}\left\lvert\frac{x_t - y_t}{x_t}\right\rvert$ | Adicional | Minimizar |

*Fuente: elaboración propia a partir de (Hewamalage, Ackermann & Bergmeir, 2022) y de (Kim, Kim, 2016).*

[6] Se recomienda la consulta de (Hewamalage, Ackermann y Bergmeir, 2022) para más detalles acerca de las métricas mencionadas. Queda fuera del ámbito de este trabajo el detalle del uso de cada una de estas.
[7] Siendo: $x_t$ el valor verdadero, $y_t$ la predicción, $SSR$ la suma de los cuadrados de los residuos y $SST$ la suma total de los cuadrados.

c. Modelos ARIMA

La primera tipología de modelos disponible para entrenar en Forecaster son los conocidos modelos estadísticos o econométricos ARIMA. Hay implementados tres métodos diferentes para obtener modelos de esta clase.

i. Auto ARIMA

La opción más sencilla y rápida para obtener modelos ARIMA consiste en la elección de los parámetros *p, d, q* sin intervención alguna del usuario. Esto se realiza a través de la función del paquete *forecast.* Este algoritmo calcula el AICc (criterio de información de Akaike corregido) para cada una de las combinaciones de los parámetros anteriores y selecciona aquella que minimice esta métrica de evaluación. La automatización determinará también si el modelo a entrenar incluye los términos estacionales de un modelo SARIMA (Hyndman y Khandakar, 2008).

ii. ARIMABoost

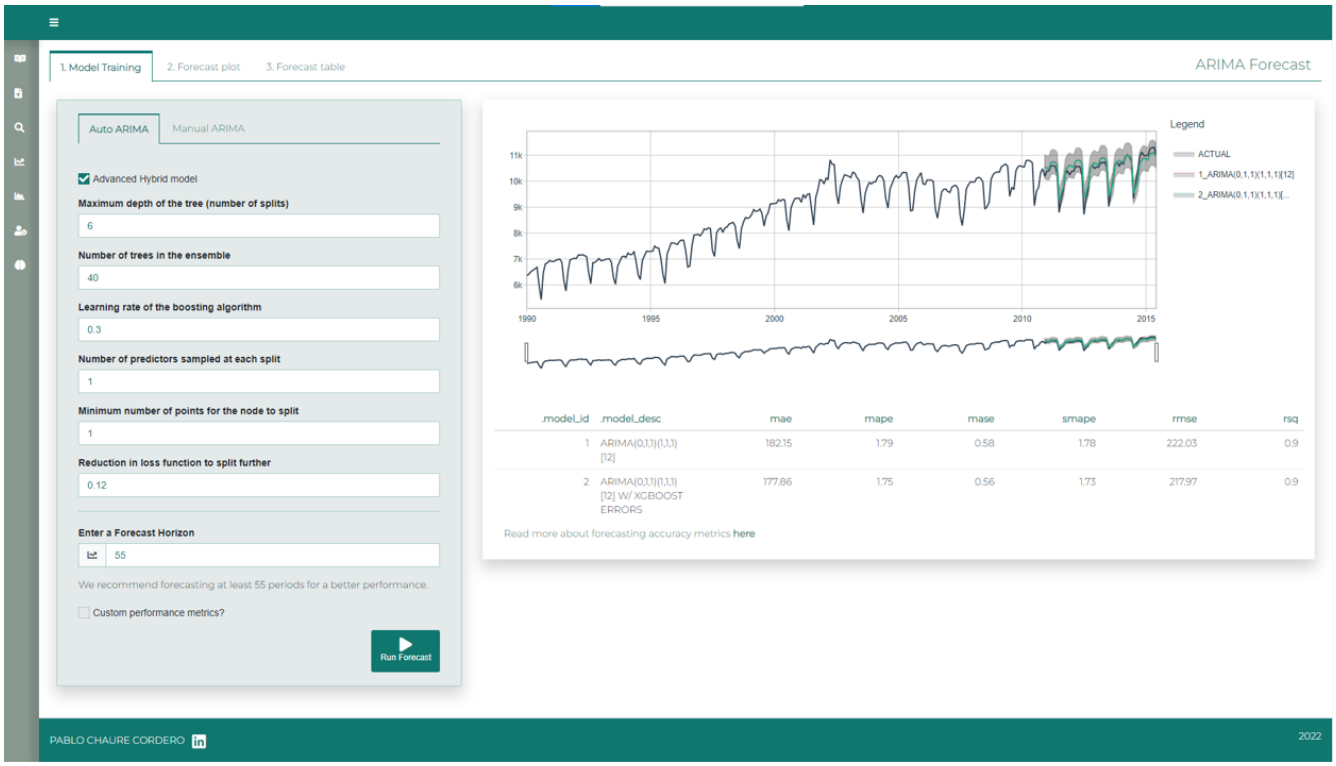
En Forecaster se ha implementado un modelo más avanzado de la especificación de ARIMA que combina las predicciones de este con la modelización de los residuos del modelo con el algoritmo de gradiente de árboles de decisión reforzados XGBoost (Blume, 2022).

El usuario puede entrenar este modelo marcando la caja de *Advanced hybrid model*. Para su entrenamiento habrá de especificar seis parámetros diferentes correspondientes al algoritmo de XGBoost. Estos son:

- *mtry*: el número de predictores que se muestrean aleatoriamente en cada *split* al crear los modelos de árbol.
- *trees*: el número de árboles del ensamblado.
- *min_n:* el número mínimo de observaciones en un nodo que se requieren para que el nodo se siga dividiendo.
- *tree_depth*: la profundidad máxima del árbol (es decir, el número de divisiones que se realizan).

- *learn_rate:* la velocidad a la que el algoritmo se adapta de iteración en iteración.
- *loss_reduction:* la reducción de la función de pérdida necesaria para seguir dividiendo el árbol.

**Figura 19:** Entrenamiento de AutoARIMA y ARIMABoost

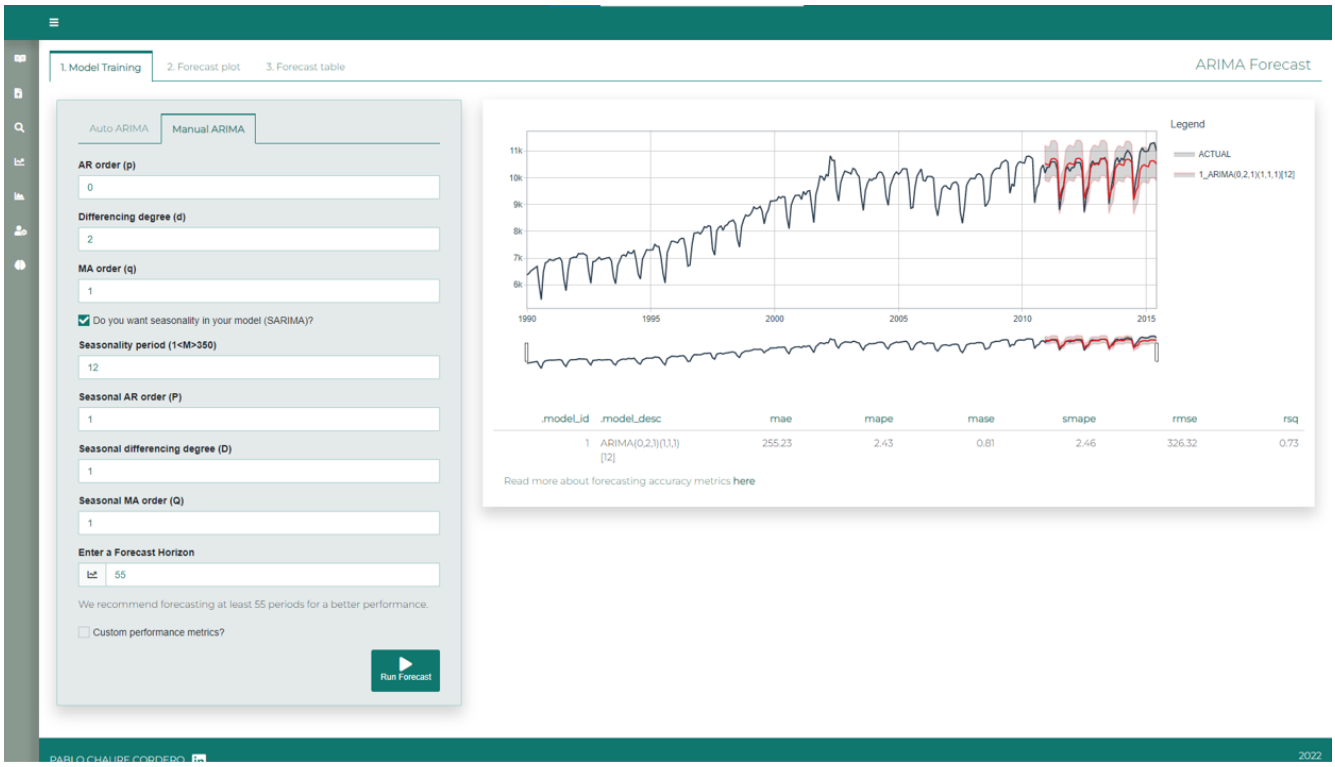



*Fuente: elaboración propia a partir de una impresión de pantalla de Forecaster*

iii. Manual ARIMA

El usuario también es capaz de entrenar un modelo ARIMA o SARIMA especificando individualmente cada uno de los parámetros que estos modelos requieren. Así, la primera elección que debe hacer es si desea añadir el componente estacional al modelo y generar un modelo SARIMA o, al no necesitarlo, continúa con un modelo ARIMA. Así, el usuario ha de especificar un máximo de siete parámetros: *p*, *d*, *q* para ARIMA; *P*, *D*, *Q*, *m* para SARIMA.

**Figura 20:** Entrenamiento manual de ARIMA




*Fuente: elaboración propia a partir de una impresión de pantalla de Forecaster*

d. Modelos ETS

A continuación, se pueden generar otros modelos estadísticos del tipo ETS o suavizado exponencial. Para estos modelos es muy relevante el análisis exploratorio correspondiente a la descomposición de la serie temporal en la estacionalidad (S), la tendencia (T) y el error (E). A través del análisis de estas componentes, el usuario es capaz de seleccionar el tipo de algoritmo de ETS que más le conviene a la serie dadas sus características. Los modelos de ETS son tres: simple, doble o de Holt; y Holt-Winters'. El usuario puede elegir entrenar todos ellos a la vez o individualmente o, también, abstenerse de especificar cuál y recurrir a un modelo completamente automatizado. Además, para cada uno de los tres modelos la especificación de cada parámetro también se puede automatizar si así se desea.

i. Auto ETS

En primer lugar, se ofrece la posibilidad al usuario de generar un modelo ETS sin la necesidad de especificar de qué tipo de modelo se trata. El modelo será de uno de los tres tipos dependiendo de si la serie tiene tendencia y/o estacionalidad definida. De este modo, el Auto ETS generará un modelo simple, de Holt o Holt-Winters' y seleccionará la mejor combinación de los parámetros para el usuario. Al igual que con Auto ARIMA, la selección de los mejores parámetros para la serie se realiza atendiendo al AICc como medida de la calidad relativa del modelo respecto a otros modelos. Se generan las combinaciones posibles de los parámetros y, aquella que genere el menor valor de AICc, se considera la mejor y se toma como definitiva, aunque esto no implica que el modelo sea necesariamente bueno y se habría de evaluar la calidad absoluta del modelo (Hyndman y Athanasopoulos, 2018).

ii. Modelo simple

El modelo simple de ETS busca predecir la serie atendiendo al nivel. Este modelo es adecuado cuando la serie no presenta una tendencia y estacionalidad evidentes. El parámetro que se puede modificar en esa componente del nivel es el error de la serie y este puede ser del tipo aditivo o multiplicativo. Por ello, el usuario tiene la posibilidad de definir si quiere entrenar el modelo con un error aditivo o multiplicativo. También puede dejar que Forecaster establezca ese valor de manera automática.

iii. Modelo doble o de Holt

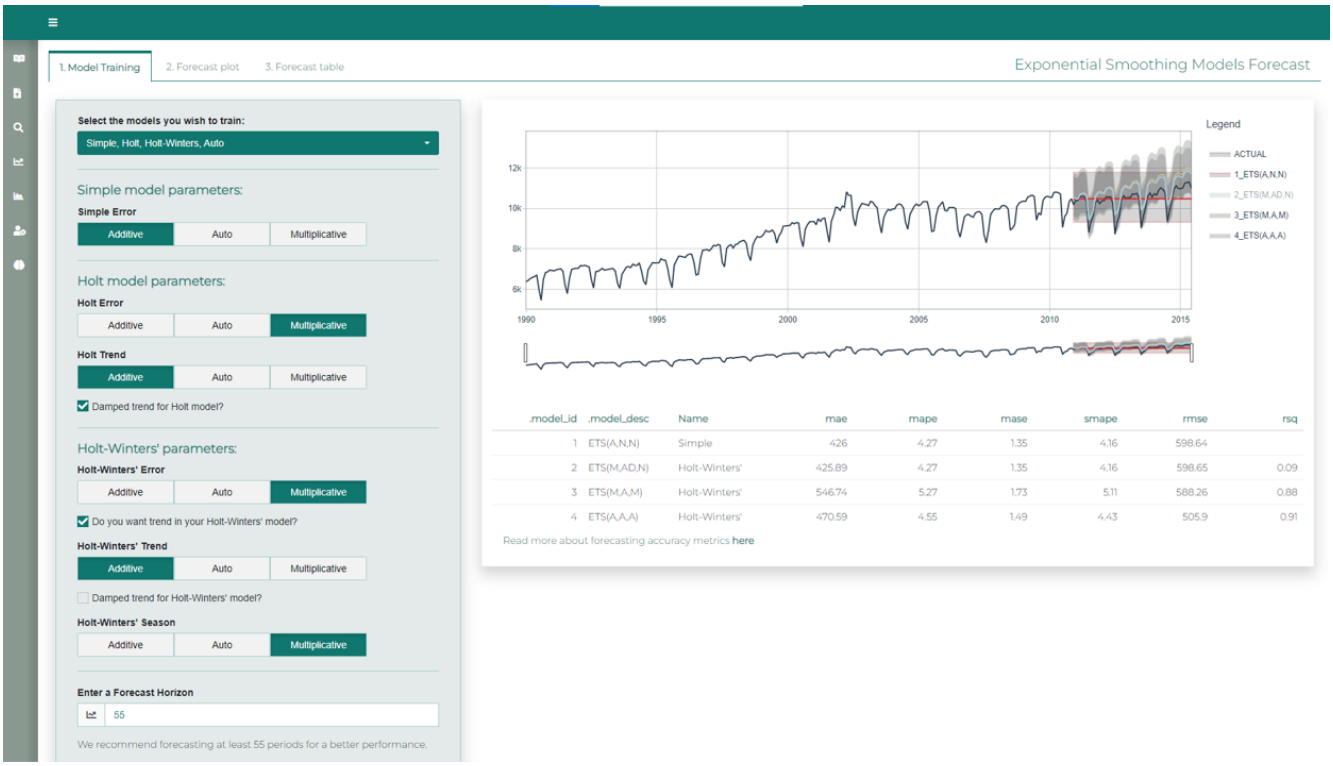
El modelo de Holt es adecuado para hacer *forecasting* cuando la serie presenta una tendencia marcada. Acumuladamente con el error como en el modelo simple, este modelo añade la componente de la tendencia que puede ser lineal o multiplicativa. Este valor es definible por el usuario a través de los botones de la UI con la opción automática, como se viene mencionando. Además, a la tendencia que se modela se le puede aplicar un factor de mitigación o *damping* que busca reducir el efecto de la tendencia a largo plazo. Si una tendencia se mantiene en el tiempo con el mismo grado es probable que acabe "sobrepredicciendo" los valores por exceso (tendencia positiva) o por defecto (tendencia

negativa) cuando el horizonte es largo. Esta opción de aplicar *damping* a la tendencia del modelo también está integrado en la UI.

### iv. Modelo de Holt-Winters'

Por último, cuando una serie presenta estacionalidad el modelo de Holt-Winters' resulta el más adecuado. Este, una vez más, acumula las componentes de nivel y tendencia junto con la estacionalidad para generar un modelo que capture todos los elementos que conforman la serie. Al igual que en los casos anteriores se ha de determinar por el usuario si los parámetros del modelo son aditivos, multiplicativos o automáticos, si se quiere incluir la componente de la tendencia (pues puede ser que la serie no la presente) y si aplica un *damping* a la tendencia.

**Figura 21:** Entrenamiento de modelos de ETS




*Fuente: elaboración propia a partir de una impresión de pantalla de Forecaster*

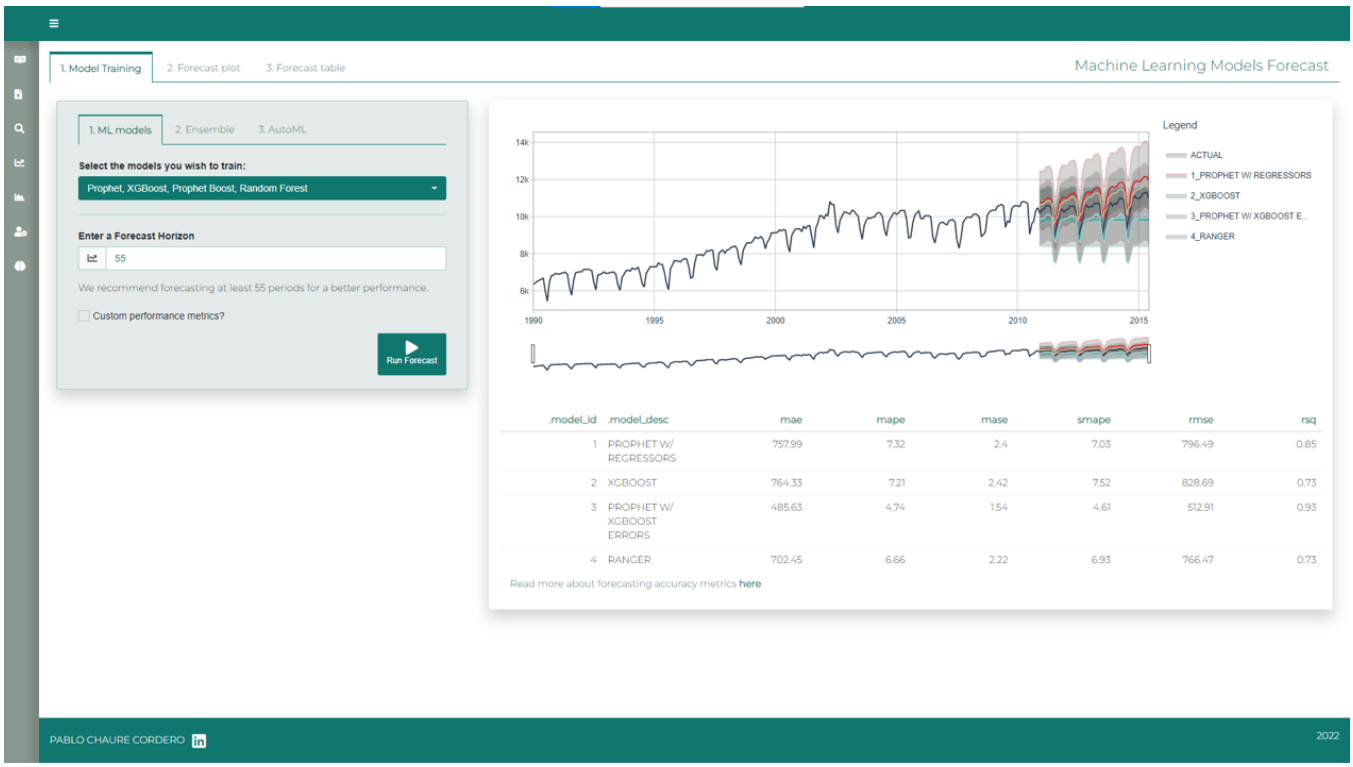
e. Modelos de Machine Learning

En la sección de ML, se pueden entrenar modelos predictivos con tres procedimientos distintos. Por un lado, se pueden entrenar modelos provenientes de cuatro algoritmos de ML de modo individual. Estos modelos entrenados previamente, a su vez pueden combinarse en modelos individuales de ensamblado con tres estrategias disponibles. Por último, está disponible toda la funcionalidad de H2O.ai para el entrenamiento de modelos de ML de forma completamente automatizada aumentando enormemente las posibilidades que esta sección ofrece.

i. Modelos individuales de ML

Forecaster ofrece algoritmos de ML con los que entrenar hasta cuatro modelos simultáneamente para una misma serie. Los algoritmos disponibles son: *Prophet* (Taylor y Letham, 2018), *XGBoost* (Chen y Guestrin, 2016), *ProphetBoost* (Ansoleaga, 2022) y *Random Forest*. Estos algoritmos admiten un número elevado de parámetros a ajustar para adaptar el algoritmo a las características de la serie. En Forecaster no está disponible la especificación de todos estos argumentos que pueden mejorar considerablemente las predicciones. Esto es así para no sobrecargar al usuario con información potencialmente abrumadora, para no abarrotar la UI con una larga lista de *inputs* y, sobre todo, para seguir siendo fiel al diseño automatizado y de experimentación que rodea a toda la aplicación. Este detalle no implica que los modelos no vayan a poder predecir bien las señales de la serie, pero lo harán en menor medida que si hubiese un ajuste específico de estas opciones. Para suplir esta merma de precisión, Forecaster tiene un preprocesamiento específico de los datos para aumentar la información que las variables independientes son capaces de aportar a la predicción. En un futuro desarrollo de la aplicación, se introducirán procesos de ajuste de hiperparámetros de los modelos con las funcionalidades que el paquete *modeltime.resample* ofrece.

**Figura 22:** Entrenamiento de modelos individuales de ML




*Fuente: elaboración propia a partir de una impresión de pantalla de Forecaster*

ii. Modelos ensamblados

Una vez los modelos de ML se han entrenado, evaluado y predicho con ellos, el usuario puede optar por generar una nueva especificación de esos modelos a través de una combinación de estos. Se ha demostrado en competiciones y en investigaciones, que el *ensembling* es una de las técnicas que mejores resultados arroja porque combina las fuerzas de cada uno de los modelos para cubrir las deficiencias de otros de los modelos participantes.

Forecaster todavía no incluye las técnicas de ensamblado por *stacking* que se han mencionado previamente, si bien es una de las técnicas próximas a desarrollar. No obstante, sí que se pueden entrenar modelos ensamblados por promedio ponderado; y ensamblados más simples por media y mediana de los modelos individuales.

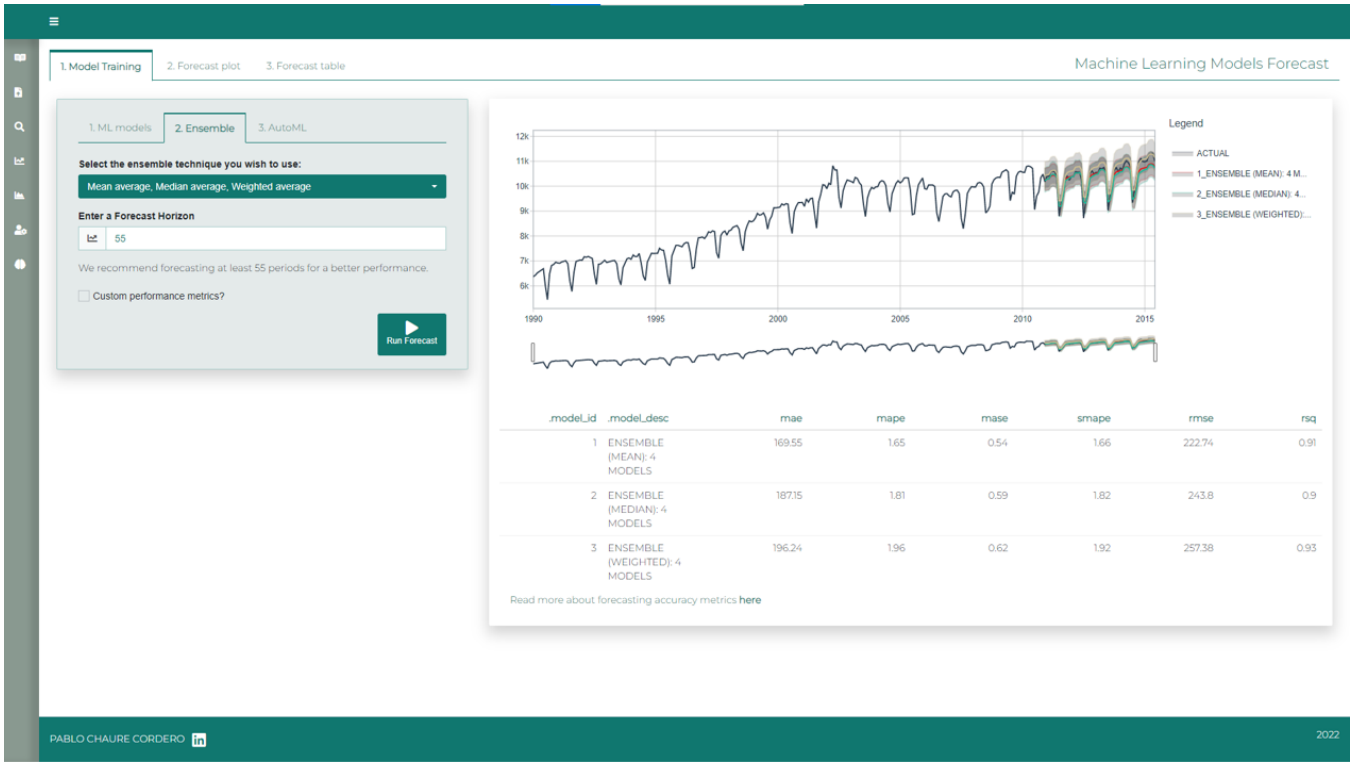
El ensamblado por media y mediana son técnicas que toman un promedio simple de las predicciones de los submodelos, ya sea por la media aritmética o por la mediana de los

modelos. Por lo tanto, todos los modelos del *ensemble* recibirán la misma carga o ponderación. Ha de tenerse en cuenta que, si se parte de modelos extremadamente malos, el rendimiento del ensamblado se verá muy afectado, puesto que estas técnicas tenderán hacia la media de modelos malos.

En segundo lugar, el ensamblado por promedio ponderado funciona de la misma manera que los anteriores, pero ahora cada modelo recibe un peso diferente. Estos pesos se basan en una simple clasificación de todos los modelos introducidos, a los que se otorga un peso relativo a su posición en dicha clasificación.

Las dos primeras técnicas son métodos rápidos y buenos, pero el ensamblado por media ponderada puede, potencialmente, obtener mejores resultados, ya que se da más valor al modelo que individualmente tiene mejor rendimiento.

**Figura 23:** Entrenamiento de modelos ensamblados




*Fuente: elaboración propia a partir de una impresión de pantalla de Forecaster*

iii. AutoML con H2O.ai

Esta función permite al usuario entrenar numerosos modelos de ML utilizando el algoritmo de H2O.ai, H2O AutoML. El entrenamiento automático de modelos de ML es el proceso de automatizar la selección de algoritmos, la generación de variables (*feature engineering*), el ajuste de hiperparámetros, el modelado iterativo y la evaluación de modelos. AutoML facilita enormemente el entrenamiento y la evaluación de los modelos de aprendizaje automático para la tarea de previsión en cuestión.

En esta sección sí que se muestran ciertos parámetros para que el usuario guíe el funcionamiento del algoritmo. Algunos de estos parámetros son: cuánto tiempo se quiere tener a la máquina entrenando modelos, el máximo de tiempo a dedicar en cada uno de los modelos, el número máximo de modelos a entrenar y si se quiere introducir un proceso de *k-fold cross validation*.

A partir de los modelos entrenados en el tiempo especificado, se seleccionan los de mejor rendimiento mediante un análisis de la matriz de modelos para el ajuste de hiperparámetros. Además, una vez entrenados todos los modelos, AutoML crea dos modelos *stacked* ensemble: uno que combina todos los modelos y otro con los mejores modelos de cada familia o tipo de algoritmos. De todos los modelos entrenados, el usuario será mostrado el que mejor evaluación presente en términos de las medidas de precisión calculadas con los residuos.

**Figura 24:** Entrenamiento de modelos con AutoML


*Fuente: elaboración propia a partir de una impresión de pantalla de Forecaster*

f. Modelos de Deep Learning

En cuarto y último lugar, Forecaster facilita el entrenamiento y la experimentación con modelos muy avanzados de DL. Actualmente hay dos algoritmos de DL disponibles en la aplicación: DeepAR y N-Beats Ensemble. Para el desarrollo de esta parte, se ha dependido en gran medida de la librería GluonTS desarrollada por Amazon Web Services para la predicción probabilística de series temporales, enfocada en algoritmos de DL.

i. DeepAR

DeepAR se trata de un algoritmo de DL del tipo RNN (*Recurrent Neural Network*). Este modelo muestra su fortaleza cuando el problema en cuestión trata de predecir numerosas series temporales de un mismo *dataset*. Es un método de *forecasting* basado en RNN autorregresivas que genera un modelo global a partir de los datos históricos de todas las series temporales con que se entrena. DeepAR combina el enfoque probabilístico para la

predicción con la inclusión de una red LSTM (*Long Short-Term Memory*) (Salinas et al., 2020).

Para el entrenamiento de modelos con DeepAR al igual que con los modelos individuales de ML, no es necesaria la especificación de todos los parámetros que definen el modelo. No obstante, como requieren los algoritmos de DL, sí que hay que especificar el número de *epochs* para determinar cuántas veces el algoritmo pasa por todo el set de entrenamiento.

ii. N-Beats Ensemble

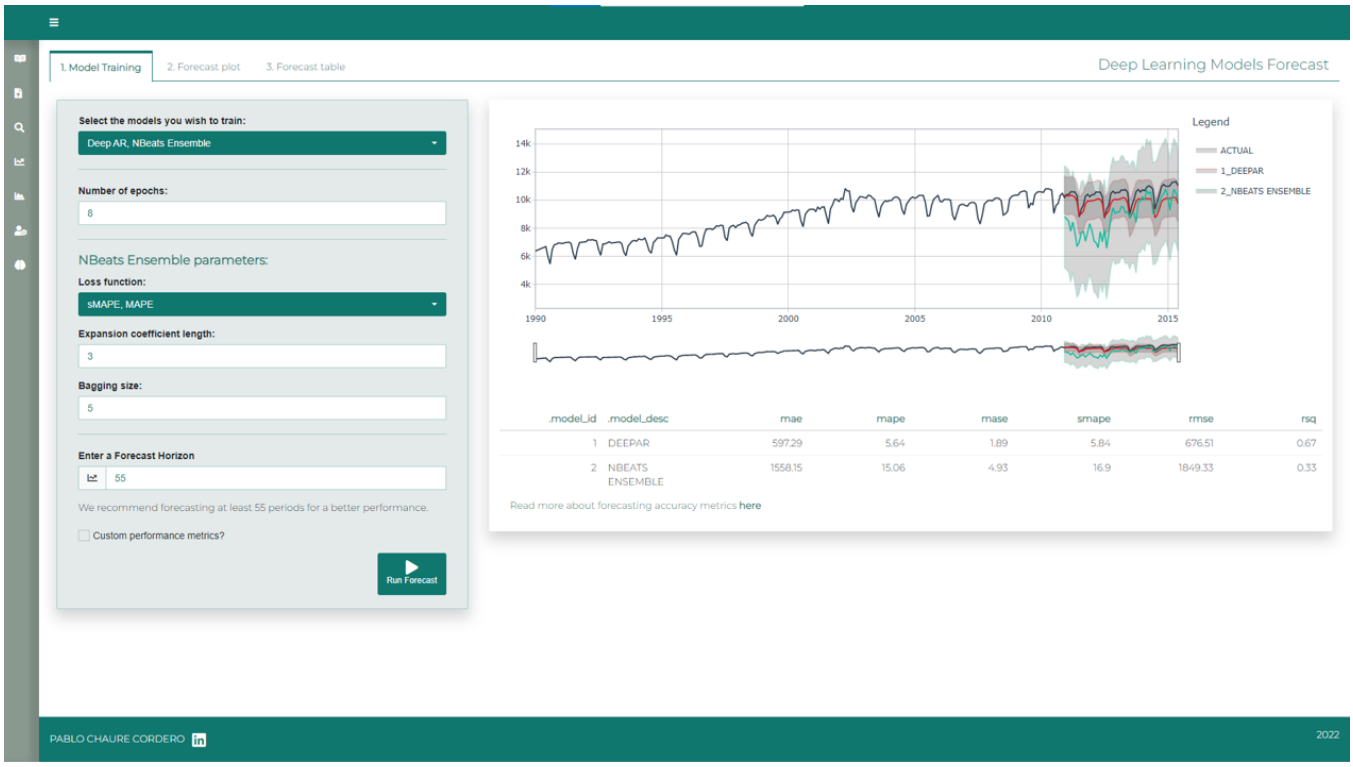
N-Beats es un algoritmo de DL basado en una arquitectura de redes neuronales profundas. Se basa en enlaces *backward* y *forward* para los residuos del modelo en una estructura muy profunda de capas de neuronas completamente conectadas. Ha sido un algoritmo que ha mostrado mejoras significativas a las técnicas ganadoras de las competiciones M3 y M4.

La especificación de ensemble del modelo busca añadir diversidad y flexibilidad al modelo entrenado. Este objetivo se consigue a partir del entrenamiento de numerosos modelos que luego forman el ensemble. El número de modelos a entrenar viene especificado por la siguiente relación:

$$|Expansion\ coefficient| * |Loss\ function| * Bagging\ size$$

Los modelos que lo conforman son entrenados en base a tres métricas diferentes (sMAPE, MASE y MAPE) para cubrir las deficiencias de cada una. Estas métricas pueden ser seleccionadas por el usuario como uno de los parámetros ajustables para este modelo. En segundo lugar, los modelos individuales son entrenados en ventanas de tiempo de longitudes variables (2*Horizonte (H), 3H, …). Por último, se lleva a cabo un procedimiento de *bagging* que incluye los modelos entrenados en inicializaciones aleatorias. Puede ser definido por el usuario a través del *bagging size*, que indica el número de modelos del ensemble que comparten las combinaciones de *loss function* y el *lookback period* (Oreshkin et al., 2019).

**Figura 25:** Entrenamiento de modelos de DL




*Fuente: elaboración propia a partir de una impresión de pantalla de Forecaster*

### *4.6.4. Productos de la aplicación*

Como se ha visto, Forecaster genera múltiples cálculos, modelos y visualizaciones a lo largo del proceso de análisis y entrenamiento de los modelos. Prácticamente la totalidad de estas salidas generadas son posibles de descargar para su aprovechamiento en otras herramientas o para mostrar los resultados obtenidos a través de otros medios.

Todas las visualizaciones que Forecaster genera y muestra son descargables en formato PNG para su publicación por otros medios. Mientras que en la aplicación son todo visualizaciones interactivas y dinámicas gracias a *plotly*, una vez se descargan se pierde esa flexibilidad puesto que el formato de salida es estático.

Por otra parte, se pueden exportar las tablas con los resultados de cada uno de los modelos entrenados para el horizonte indicado. El formato de salida es un archivo CSV nombrado con la siguiente estructura:

{horizonte}_{tipología del modelo}_forecast_{nombre del archivo de entrada}.csv

Este incluye la fecha, un identificador y una descripción del modelo en cuestión; el valor futuro en la mediana y los valores futuros en los extremos del intervalo de confianza. Este archivo puede ser aprovechado por el usuario para su introducción en otros sistemas que ayuden a la toma de decisiones.

## 4.7. Futuros desarrollos

Forecaster es una herramienta que busca reunir en un único lugar las técnicas más avanzadas que se van desarrollando en el ámbito de la predicción y el análisis de las series temporales. Como se ha ido introduciendo a lo largo del trabajo, existen ciertas áreas, tecnologías y métodos que Forecaster todavía no cubre para lograr ser una herramienta de experimentación realmente completa. Por este motivo, la herramienta que en este trabajo se presenta es únicamente una primera versión de lo que se quiere que Forecaster llegue a ser. Durante el desarrollo de Forecaster se han ido identificando nuevos módulos, análisis, modelos y técnicas que se quieren incluir en la herramienta.

En cuanto a modelos adicionales que se quieren incluir se encuentran modelos muy simples, pero a veces convenientes como los modelos *naive* y también otros más avanzados como TBATS y STLM, modelos de regresión de múltiples estacionalidades. Las secciones de ML y DL pueden ser desarrolladas con la inclusión de muchos modelos que se van adaptando a los problemas con series temporales. Junto con la inclusión del *stacking* para el ensamblado de modelos, el objetivo está puesto en disponer de más de veinte técnicas y modelos disponibles en estas tipologías de modelos. Adicionalmente, como se ha anticipado, se quiere incluir un ajuste automatizado de hiperparámetros en el proceso interno de entrenamiento de los modelos.

Las capacidades analíticas de Forecaster quieren ser ampliadas con la inclusión y cálculo de tests estadísticos como el test aumentado de Dickie Fuller y otros test de estacionariedad; y el análisis de los residuos de los modelos. Además, para entender mejor los grupos de series temporales o cuándo estas se presentan de modo jerárquico, se está explorando la inclusión de una sección para hacer *clustering* de las series que ayude en la generación de modelos para múltiples series a la vez.

Fuera de estas categorías, se quiere implementar un módulo que genere un reporte interactivo y dinámico que muestre visual y analíticamente todos los modelos entrenados y las visualizaciones generadas para la serie en cuestión. Por último, se desea migrar el despliegue de la aplicación de Shinyapps.io a una máquina virtual alojada en un servidor Linux en la nube y la generación de contenedores para aportar un mejor rendimiento de la aplicación cuando la afluencia de usuarios es más elevada.

# 5. CONCLUSIÓN

Forecaster se postula como una herramienta útil, eficaz y de valor para llevar a cabo la tarea de análisis y predicción de series temporales por parte de científicos de datos y analistas de negocio. Estos roles de la organización no son quienes tienen la última palabra en la función de planificación y control. Esta corresponde a los directivos y empresarios encargados del desarrollo del negocio y de tomar, en última instancia, las decisiones que afectan al mismo. De este modo, con las capacidades de análisis y experimentación de modelos de predicción para datos de serie de tiempo que posee Forecaster, se puede considerar como una adición valiosa al conjunto de servicios tecnológicos que una empresa debe considerar.

La mayor aspiración que tiene Forecaster es la de postularse como una herramienta con la que facilitar la tarea de *forecasting* con la variedad de modelos que reúne, siendo flexible y apta para todos los casos de uso con los que una organización se pueda topar. Para cumplir con este objetivo, Forecaster debe ser actualizada y desarrollada continuamente para poder incorporar y ofrecer todas las técnicas y la tecnología más novedosa y avanzada para lograr los resultados más precisos posibles. Así, se recomienda seguir de cerca los avances de esta área de la ciencia de datos, especialmente aquellos de las Competiciones-M por el prestigio y relevancia que llevan teniendo desde sus inicios y que las convierte en el referente de la evolución de la predicción de series temporales.

Por la motivación anterior, Forecaster se ha definido como un proyecto de código abierto y *open source* para que cualquier persona de la comunidad pueda aportar su contribución al proyecto a través del repositorio público de GitHub. Con la incorporación de nuevos módulos, modificaciones y correcciones del actual software, Forecaster puede llegar a ser más completa, robusta y se eliminan deficiencias y debilidades que puedan existir.

Este escrito sirve como documentación técnica de Forecaster y puede ser utilizado como referencia para dichos desarrolladores o usuarios. En el caso de los usuarios también es útil en la medida en que se describe la estructura y el flujo de trabajo dentro de la aplicación, haciéndolo una fuente relevante para profundizar en el contenido y uso de Forecaster.

En definitiva, Forecaster supone una herramienta novedosa y completa dentro de las alternativas que uno es capaz de encontrar en el mercado. Ofrece amplias posibilidades y flexibilidad para la tarea que trata de acometer a la vez de ser completamente gratuita. Con el continuo desarrollo de paquetes y librerías como modeltime, se puede llevar esta práctica de la ciencia de datos al siguiente nivel haciéndolo accesible al público general. Sería interesante acometer, de manera colaborativa, los evolutivos detallados en la última sección de futuros desarrollos para llevarla un paso más allá y convertirla en el líder de este campo dentro del ecosistema *open source.*

# 6. BIBLIOGRAFÍA


*CRAN Task Views* 2022, [Homepage of Comprehensive R Archive Network (CRAN)], [Online]. Disponible en: https://CRAN.R-project.org/web/views/.

*What is Three-Tier Architecture*. 2021. Disponible en: www.ibm.com/cloud/learn/three-tier-architecture.

*Shiny - Reactivity. An overview.* 2017. Disponible en: https://shiny.rstudio.com/articles/reactivity-overview.html.

*What is a Web-Based Application?* 2017. Disponible en: http://www.techopedia.com/definition/26002/web-based-application.

*What is a Framework? Why We Use Software Frameworks* 2015, Code Institute.

*Shiny*. Disponible en: https://www.rstudio.com/products/shiny/.

*What is Shiny (in R)?*. Disponible en: https://www.dominodatalab.com/data-science-dictionary/shiny-in-r.

*What is User Experience (UX) Design?* Disponible en: https://www.interaction-design.org/literature/topics/ux-design.

Alegsa, L. 2018, *Definición de aplicación web*. Disponible en: https://www.alegsa.com.ar/Dic/aplicacion_web.php.

Ansoleaga, U.L. 2022, *Boost Your Time Series Forecasts Combining Gradient Boosting Models with Prophet Features*. Disponible en: https://towardsdatascience.com/boost-your-time-series-forecasts-combining-gradient-boosting-models-with-prophet-features-8e738234ffd.

Blume, T. 2022, *Gradient Boosted ARIMA for Time Series Forecasting*. Disponible en: https://towardsdatascience.com/gradient-boosted-arima-for-time-series-forecasting-e093f80772f6.

Box, G.E. 1979, "Robustness in the strategy of scientific model building" in *Robustness in statistics.* Elsevier, pp. 201-236.

Bustos, G. 2021, *¿Qué es localhost?*, Hostinger.

Cetinkaya-Rundel, M. 2017, *Shiny - Two-file Shiny apps*. Disponible en: https://shiny.rstudio.com/articles/two-file.html.

Chang, W. 2017, *Shiny - App formats and launching apps*. Disponible en: https://shiny.rstudio.com/articles/app-formats.html.

Chang, W. y Borges Ribeiro, B. 2018, *shinydashboard: Create Dashboards with 'Shiny'*.

Chen, T. y Guestrin, C. 2016, "Xgboost: A scalable tree boosting system", *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785.

Dancho, M. 2021, *Package 'modeltime.h2o'*, EEUU.

Dancho, M. 2020, *Introducing Modeltime: Tidy Time Series Forecasting using Tidymodels*. Disponible en: https://www.business-science.io/code-tools/2020/06/29/introducing-modeltime.html.

Dancho, M., *GluonTS Deep Learning*. Disponible en: https://business-science.github.io/modeltime.gluonts/index.html.

Dancho, M., *Installation Guide*. Disponible en: https://business-science.github.io/modeltime.gluonts/articles/managing-envs.html

Dancho, M., *Modeltime H2O Machine Learning*. Disponible en: https://business-science.github.io/modeltime.h2o/index.html

Dancho, M., *Timetk: A Tool Kit for Working with Time Series in R*. Disponible en: https://business-science.github.io/timetk/.

Dancho, M., *Visualize Anomalies for One or More Time Series — plot_anomaly_diagnostics*. Disponible en: https://business-science.github.io/timetk/reference/plot_anomaly_diagnostics.html.

Dancho, M., *Visualize the ACF, PACF, and CCFs for One or More Time Series. plot_acf_diagnostics*. Disponible en: https://business-science.github.io/timetk/reference/plot_acf_diagnostics.html.

Escalona, M.J. y Koch, N. 2002, "Ingeniería de Requisitos en Aplicaciones para la Web–Un estudio comparativo", *Universidad de Sevilla*.

Finzer, W. 2013, "The data science education dilemma", *Technology Innovations in Statistics Education*, vol. 7, no. 2.

Gourley, D., Totty, B., Sayer, M., Aggarwal, A. y Reddy, S. 2002, *HTTP: the definitive guide*, " O'Reilly Media, Inc.".

Grolemund, G. 2015, *Shiny - How to understand reactivity in R*. Disponible en: https://shiny.rstudio.com/articles/understanding-reactivity.html.

Hewamalage, H., Ackermann, K. y Bergmeir, C. 2022, "Forecast Evaluation for Data Scientists: Common Pitfalls and Best Practices", *arXiv preprint arXiv:2203.10716*.

Hyndman, R.J. y Athanasopoulos, G. 2018, *Forecasting: Principles and Practice (2nd ed)*, Segunda ed., OTexts, Melbourne, Australia.

Hyndman, R.J. y Khandakar, Y. 2008, "Automatic time series forecasting: the forecast package for R", *Journal of statistical software*, vol. 27, pp. 1-22.

Kim, S. y Kim, H. 2016, "A new metric of absolute percentage error for intermittent demand forecasts", *International Journal of Forecasting*, vol. 32, no. 3, pp. 669-679.

Kuhn, M., *Explore tidymodels - Search parsnip models*. Disponible en: https://www.tidymodels.org/find/parsnip/.

Lilovski, N. 2021, *dashboardthemes: Customise the Appearance of 'shinydashboard' Applications using Themes.*

Luján-Mora, S. 2002, *Programación de aplicaciones web: historia, principios básicos y clientes web*, Editorial Club Universitario.

Makridakis, S., Spiliotis, E. y Assimakopoulos, V. 2020, "The M4 Competition: 100,000 time series and 61 forecasting methods", *International Journal of Forecasting*, vol. 36, no. 1, pp. 54-74.

Mauricio, J. 2018, *Understanding Web Development Frameworks in 2018*, Level Up Coding.

Moraga, P. 2019, *Geospatial health data: Modeling and visualization with R-INLA and shiny*, CRC Press.

Oreshkin, B.N., Carpov, D., Chapados, N. & Bengio, Y. 2019, "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting", *arXiv preprint arXiv:1905.10437*.

R Core Team 2021, *R: A language and environment for statistical computing*, Viena, Austria.

Salinas, D., Flunkert, V., Gasthaus, J. y Januschowski, T. 2020, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks", *International Journal of Forecasting*, vol. 36, no. 3, pp. 1181-1191.

Taylor, S.J. y Letham, B. 2018, "Forecasting at scale", *The American Statistician*, vol. 72, no. 1, pp. 37-45.

Torres, M. 2018, *Pasos para diseñar una aplicación web exitosa*.

Ushey, K., Tang, Y. y Allaire, J.J. 2022, *reticulate: Interface to 'Python'*.

Van Rossum, G. y Drake, F.L. 2018, *Python 3 Reference Manual*, CA, EEUU.